Blazor, Python, iOS, ChatGPT, Stable Diffusion

CODE

MAR
APR
2025

codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 8.95 Can $ 11.95

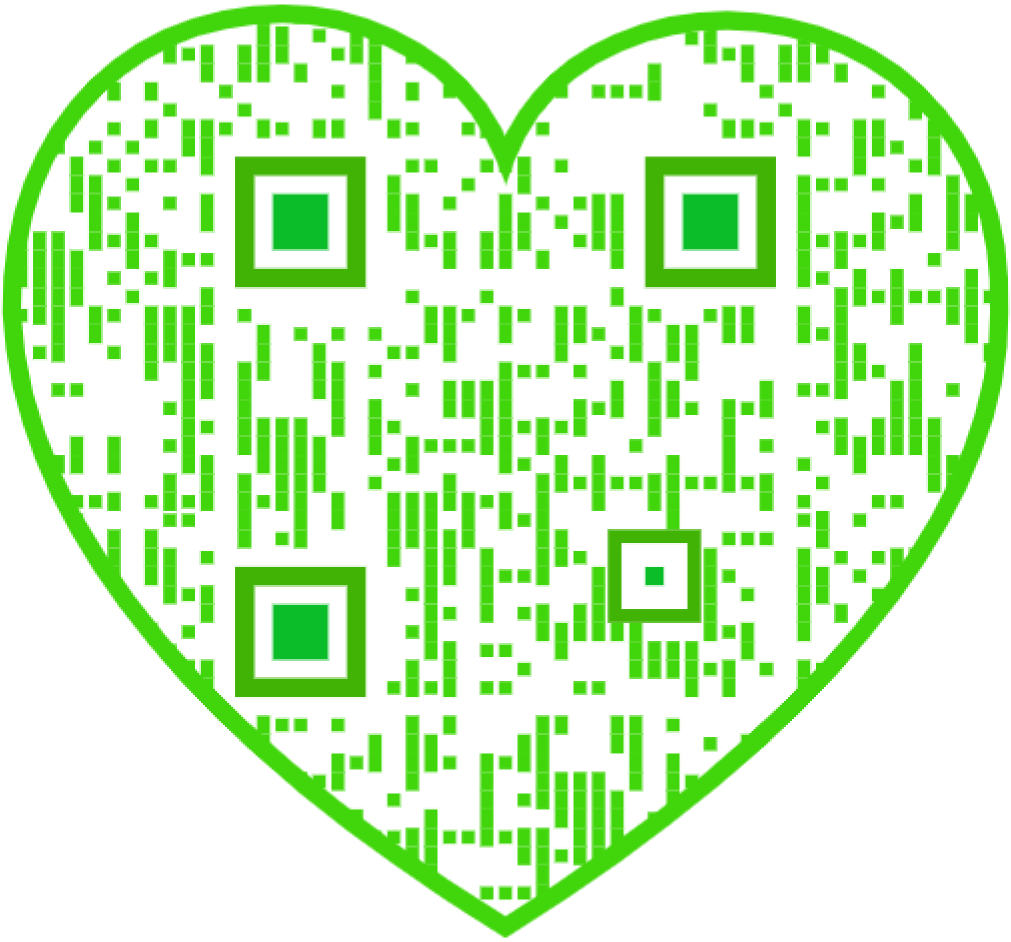
Cover AI generated - Markus Egger

Building Applications with Blazor and ASP.NET

Working with Lists in .NET MAUI

Exploring ChatGPT with iOS

Offline AI Image Generation



CODE
CONSULTING

ARE YOU WONDERING HOW ARTIFICIAL INTELLIGENCE CAN BENEFIT YOU TODAY?

EXECUTIVE BRIEFINGS

Are you wondering how AI can help your business? Do you worry about privacy or regulatory issues stopping you from using AI to its fullest? We have the answers! Our Executive Briefings provide guidance and concrete advise that help decision makers move forward in this rapidly changing Age of Artificial Intelligence and Copilots!

We will send an expert to your office to meet with you. You will receive:

1. An overview presentation of the current state of Artificial Intelligence.
2. How to use AI in your business while ensuring privacy of your and your clients' information.
3. A sample application built on your own HR documents – allowing your employees to query those documents in English and cutting down the number of questions that you and your HR group have to answer.
4. A roadmap for future use of AI catered to what you do.

AI-SEARCHABLE KNOWLEDGEBASE AND DOCUMENTS

A great first step into the world of Generative Artificial Intelligence, Large Language Models (LLMs), and GPT is to create an AI that provides your staff or clients access to your institutional knowledge, documentation, and data through an AI-searchable knowledgebase. We can help you implement a first system in a matter of days in a fashion that is secure and individualized to each user. Your data remains yours! Answers provided by the AI are grounded in your own information and is thus correct and applicable.

COPILOTS FOR YOUR OWN APPS

Applications without Copilots are now legacy!

But fear not! We can help you build Copilot features into your applications in a secure and integrated fashion.

CONTACT US TODAY FOR A FREE CONSULTATION AND DETAILS ABOUT OUR SERVICES.

codemag.com/ai-services
832-717-4445 ext. 9 • info@codemag.com

©shutterstock

Features

8 Exploring .NET MAUI: Working with Lists of Data

In this fifth entry in his series on MAUI, Paul teaches you how to navigate from list items to detail pages using ListView, CollectionView, and CarouselView controls.

Paul Sheriff

26 Offline AI Image Generation

If you've been fascinated by those AI-generated images (that sometimes get things wrong), you'll want to learn how Sahil uses text and image input to come up with interesting results.

Sahil Malik

38 Building Modern Web Applications Using Blazor ASP.NET Core

Blazor has been used to create interactive web apps since .NET 5 came out. Joydip shows you how flexible it is using C# and .NET to build a web app.

Joydip Kanjilal

56 Boost Your iOS App Development with ChatGPT AI-Assistance

You've probably been playing with ChatGPT, trying to stump it or see how accurate it can be. Kevin challenges it to help develop an app. You'll want to see what happens!

Kevin J. McNeish

66 Managing Diverse Data Types in a Dataset with COLUMNTRANSFER

Follow along as Gaurav explores datasets with multiple feature types for machine learning.

Gaurav Mittal

70 Unleashing the Power of GitHub Copilot in Visual Studio Code

In recent years, AI has revolutionized the way we code. One of the most exciting developments in this field is GitHub Copilot, an AI-powered code assistant developed by GitHub in collaboration with OpenAI.

John Miller

Departments

6 Editorial

24 CODE Staffing: Career Development and Staffing Reinvented

36 CODE Magazine Presents: The State of AI Mini Conference Tour

34 Advertisers Index

74 Code Compilers

US subscriptions are US $29.99 for one year. Subscriptions outside the US pay $50.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards are accepted. Back issues are available. For subscription information, send e-mail to subscriptions@codemag.com or contact Customer Service at 832-717-4445 ext. 9.

Subscribe online at www.codemag.com

CODE Component Developer Magazine (ISSN # 1547-5166) is published bimonthly by EPS Software Corporation, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A. POSTMASTER: Send address changes to CODE Component Developer Magazine, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A.

CODE CONSULTING

APPS WITHOUT COPILOTS ARE NOW LEGACY!

Microsoft has introduced Copilot-driven applications and announced that all Microsoft applications will follow this approach going forward. This aims to capitalize on the extreme productivity gains this new paradigm promises. The same applies to your own applications and systems.

We can help you build Copilots into your own applications.atered to what you do.

VISIT OUR COPILOT PAGE TO FIND OUT MORE ABOUT OUR SERVICES AND TO SEE DEMONSTRATIONS AND EXAMPLES!

codemag.com/copilot
832-717-4445 ext. 9 • info@codemag.com

©shutterstock

Software Finds Its Way

Another week, another pair of large maintenance tasks. The first task was to do a full replacement of a logging system. This logger was built over the span of 10 years, was a bit "crufty," and lacked some functionality we needed. The other project was/is a bit larger and we are code naming this one: "Changing the engine while the truck is driving down the road." I'll talk about both of these below.

When this application (The Truck) was built, its sole purpose was to maintain a client-facing portal. This included building mundane features like user and group management, menu management, entitlements, and some other minor features. The client-facing portal eventually went away, but the admin portal stayed. It was extended to manage data warehouses, data flow processes, logging, monitoring, and some reporting. This acorn of a software project became a full featured oak tree of an application.

Here's a fact: You will spend more time in maintenance than in development.

When I was new to this business, it was drilled into me that the real cost of software was in the maintenance, not the initial construction. There are some studies that attempt to quantify the cost of maintenance vs. development. The article: "Which Factors Affect Software Maintenance Costs More?" from the National Library of Medicine (https://pmc.ncbi.nlm.nih.gov/articles/PMC3610582/), mentions a number as high as 90%. It's my experience that this number doesn't seem entirely unreasonable.

Sometimes software maintenance requires a new direction. While working on a compiler issue in a .NET MAUI application we built, I raised an internal red flag on this go-around. It seems that nearly every month, we'd been fighting compiler issues because of some change made to one part or another of our development stack. Potential issue vectors included Visual Studio, .NET Core, .NET MAUI, and a few libraries. It seems like every month, we faced some complier issue or another and wasted an inordinate amount of time just getting our code to build.

What was the source of this? In all reality, it came from how we used the Roslyn compiler. We used a designer to build forms. This designer calls a code generator, which builds and compiles a generated Model-View-ViewModel (MVVM) in the runtime. Last weekend, I wasn't looking forward to another day in compiler land, so I decided to rethink how we implemented it and did some experiments to see if a new direction was viable. After a couple of hours on the weekend and a full week day, I realized that a new direction was 100% doable. But would a new direction have any added benefits or just be a lateral move? As I coded up the new direction, I began to see easier solutions to thorny problems that remained, as well as improvements to current processes that this change would enable.

The theme among these items is this: Software is a long game and can be expensive to maintain.

As you'll recall, one project was to overhaul the logging code in our admin portal. How did we do this in a live running system? The first task was to build the actual logger itself, including the viewer. Once we were happy with our code/design, we wired it into a few places that see a lot of logging activity. We let it cook for a few weeks, optimizing the logger as well as the viewer. Having real-world use helped flesh out the features with low risk to our users.

I think using polymorphism is one of the most useful techniques for lowering long-term maintenance costs. Writing to an interface can lower costs when that system may need updating or replacement. The .NET MAUI application I mentioned above needed new directions because of the constant compiler issues. The question was: Can we replace the compiler part of the application with something else? If so, how much effort would it take?

Luckily for us, this application constructs two different artifacts: a View, which is bound to properties on a View Model. Could we just swap a new type of View Model in place of the old one? After a half-day of work, the proof of concept worked and we re-implemented a few more critical features to solidify the viability of the project. After proving its viability, we made the GO decision to complete the swap.

One of my all-time favorite features of .NET is the concept of overloads. You can have a function with the same name and varying argument signatures. This is very useful in maintaining code. Function arguments are one of those "don't know what you don't know" kind of maintenance issues. Your initial pass at a function may not have the proper parameters needed, or a new requirement might need a bit more information. Bring in the wrapper. The code snippet below demonstrates this:

```
public void ReloadVendorFiles(Packet
   launchPacket)
{
ReloadVendorFiles(
   launchPacket.StartDate,
   launchPacket.EndDate,
   launchPacket.LoadType,
   launchPacket.Database);
}
public void ReloadVendorFiles(
  string startDateRange,
  string endDateRange,
  string loadType,
  string database = "THE_OTHER_DATABASE")
```

This technique involves creating a version of your function that requires additional arguments and wrapping that call inside of a simpler function, normally with lesser arguments.

With the cost of maintenance being so high, it makes sense to use techniques to help lower costs whenever possible. The techniques (new direction, pilot code, polymorphism, and wrapping/overloading functions) above are just the tip of the iceberg in making software easier to maintain. What techniques do you use?

Rod Paddock

CODE
CONSULTING

ARE YOU WONDERING HOW ARTIFICIAL INTELLIGENCE CAN HELP YOUR BUSINESS?

Do you worry about privacy or regulatory issues stopping you from using AI to its fullest?
We have the answers!

We will send an expert to your office to meet with you. You will receive:

1. An overview presentation of the current state of Artificial Intelligence.
2. How to use AI in your business while ensuring privacy of your and your clients' information.
3. A sample application built on your own HR documents – allowing your employees to query those documents in English and cutting down the number of questions that you and your HR group have to answer.
4. A roadmap for future use of AI catered to what you do.

CONTACT US TODAY FOR A FREE CONSULTATION AND DETAILS ABOUT OUR SERVICES.

codemag.com/executivebriefing
832-717-4445 ext. 9 • info@codemag.com

©shutterstock

ONLINE QUICK ID 2503021

Exploring .NET MAUI: Working with Lists of Data

In this article series, you've created several .NET MAUI pages, created a top-level menu system, and programmatically navigated between pages. Using data binding greatly reduces the amount of code you need to write. Using the MVVM and DI design patterns helps you create applications that are reusable, maintainable, and testable. In this article, you'll display lists of data and navigate from a list item to the detail page for that item. NET MAUI provides ListView, CollectionView, and CarouselView controls for displaying lists. Each list control is illustrated, and you're provided with guidance on what each control is best at displaying.

Paul D. Sheriff
http://www.pdsa.com

Paul has been working in the IT industry since 1985. In that time, he has successfully assisted hundreds of companies' architect software applications to solve their toughest business problems. Paul has been a teacher and mentor through various mediums such as video courses, blogs, articles and speaking engagements at user groups and conferences around the world. Paul has multiple courses in the www.pluralsight.com library (https://bit.ly/3gvXgvj) and on YouTube.com (https://www.youtube.com/@pauldsheriff) on topics ranging from C#, LINQ, JavaScript, Angular, MVC, WPF, XML, jQuery, and Bootstrap. Contact Paul at psheriff@pdsa.com.

Display a List of Users Using a ListView

Now that you've learned about the standard controls for input, let's turn your attention to working with lists of data. You've already seen the Picker control for displaying a small set of data. Other list controls are ListView, CollectionView, and CarouselView. The **ListView** control displays a scrolling vertical list of items. You define what each item looks like by creating a **DataTemplate** within which you use a **Cell**. A DataTemplate is a set of Cells used to display each record within your collection. As the list control iterates over your data collection, each record is displayed according to the control layout within the DataTemplate.

There are three **Cell** types used most often in a DataTemplate to define how the list of data looks. A **TextCell** displays two pieces of information on each row of the ListView: a piece of large text is set using the **Text** property, then, on the next line, a piece of smaller text is set using the **Detail** property. An **ImageCell** adds an **ImageSource** property while still using the **Text** and **Detail** properties to display large and small text next to the image. Finally, **ViewCell** allows complete flexibility to define the look for each element displayed in the list. I prefer to use ViewCell as it provides the most flexibility.

Modify the User View Model to Get a List of Users

When using .NET MAUI or WPF, always use the **ObservableCollection<T>** class for all your lists, as opposed to a List<T>. The ObservableCollection<T> object raises the appropriate notifications when the list changes. If an item is added, updated, or removed from the list, or if the whole list is recreated, any bound objects to this collection are informed and redisplay themselves using the new information in the collection.

Before you can display a list of users, you first need to modify the UserViewModel class to retrieve a list of users from the UserRepository. Open the **ViewModelClasses\UserViewModel.cs** file and add a new private property to hold a collection of users.

```
private ObservableCollection<User> _Users = new();
```

Add a new public property named **Users** to hold the list of users and to give a public collection property to bind to on the user list page.

```
public ObservableCollection<User> Users {
  get { return _Users; }
  set {
    _Users = value;
    RaisePropertyChanged(nameof(Users));
  }
}
```

Locate the GetAsync() method and modify it to use the UserRepository class to retrieve a list of users, as shown in **Listing 1.** Check to ensure that the Repository variable is not equal to null. If it's not null, call the GetAsync() method on the Repository class to return the set of users and assign that collection to the **Users** property. Set the **RowsAffected** property to the **Count** of the users. Also set the **InfoMessage** property to inform the user how many rows of users were found. Later, you'll add a label to display this informational message.

Using the TextCell

Let's modify the user list page to display a list of users using a ListView control and a TextCell. Open the **Views\UserListView.xaml** file and add a few XML namespaces to the partial views' namespace, the view model namespace, and the entity layer namespace.

```
xmlns:partial="clr-namespace:
  AdventureWorks.MAUI.ViewsPartial"
xmlns:vm="clr-namespace:
  AdventureWorks.MAUI.MauiViewModelClasses"
xmlns:model="clr-namespace:
  AdventureWorks.EntityLayer;
    assembly=AdventureWorks.EntityLayer"
```

Remove the entire <VerticalStackLayout> element and replace it with the code shown in **Listing 2.** In the ListView control, set the **SeparatorColor** and **SeparatorVisibility** properties to provide separation between each item on the Android and iOS platforms. Set the **ItemsSource** property of the ListView to the **Users** collection property of the view model. Create a <ListView.ItemTemplate> element that encloses the <DataTemplate> element. Add the x:DataType attribute on the DataTemplate starting tag to inform the Cell what type of object is expected for each row of the collection. Within the DataTemplate, add a <TextCell> element with the **Text** and **Detail** properties set to bind to the appropriate properties in the User class.

Open the **Views\UserListView.xaml.cs** file and replace the entire contents of the file with the code shown in **Listing 3.** The constructor for this class needs to have the UserViewModel injected. Assign the instance of the view model class passed into the private **_ViewModel** variable. In the OnAppearing() method set the **BindingContext** property of the ContentPage to the **_ViewModel** variable. Finally, call the GetAsync() method on the UserViewModel class to populate the **Users** property so the ListView control can bind to the list of users.

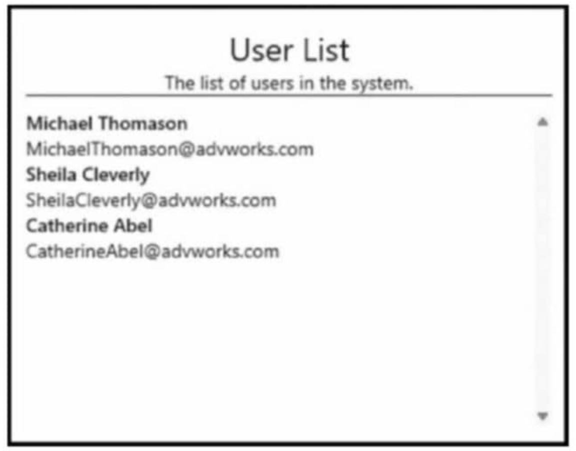
Open the **ExtensionClasses\ServiceExtensions.cs** file and add the UserListView page as a new service to the DI container. This page needs to participate in DI to have the UserViewModel injected into its constructor. Add the following line of code to the AddViewClasses() method.

```
services.AddScoped<UserListView>();
```

Try It Out

Run the application on Windows and click on the **Users** menu to see the list of users, as shown in **Figure 1.** Notice that there's no separator displayed on a Windows computer. For whatever reason, Microsoft chose not to respect the **SeparatorColor** and **SeparatorVisibility** properties on a Windows computer. Thus, the TextCell is not how I would choose to display data on a ListView. Instead, you should use a ViewCell as that allows you to keep the interface consistent between all platforms.

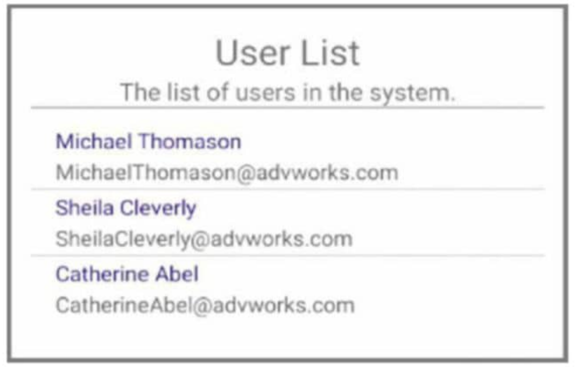
Run the application on the Android emulator and you should see that there's a separator displayed, as shown in **Figure 2.** On both Android and iOS platforms, both the **SeparatorColor** and **SeparatorVisibility** properties are respected.



User List
The list of users in the system.

Michael Thomason
MichaelThomason@advworks.com
Sheila Cleverly
SheilaCleverly@advworks.com
Catherine Abel
CatherineAbel@advworks.com

Figure 1: A ListView control using a TextCell for displaying user information on Windows



User List
The list of users in the system.

Michael Thomason
MichaelThomason@advworks.com
Sheila Cleverly
SheilaCleverly@advworks.com
Catherine Abel
CatherineAbel@advworks.com

Figure 2: A ListView control using a TextCell for displaying user information on Android

Listing 1: Retrieve a list of users from the UserRepository class and set the Users property in the view model.

```
#region GetAsync Method
public async
  Task<ObservableCollection<User>> GetAsync() {
  RowsAffected = 0;

  try {
    if (_Repository == null) {
      LastErrorMessage = REPO_NOT_SET;
    }
    else {
      Users = await _Repository.GetAsync();
      RowsAffected = Users.Count;
      InfoMessage =
        $"Found {RowsAffected} Users";
    }
  }
  catch (Exception ex) {
    PublishException(ex);
  }

  return Users;
}
#endregion
```

Listing 2: Create a ListView with a TextCell to display all users in a list.

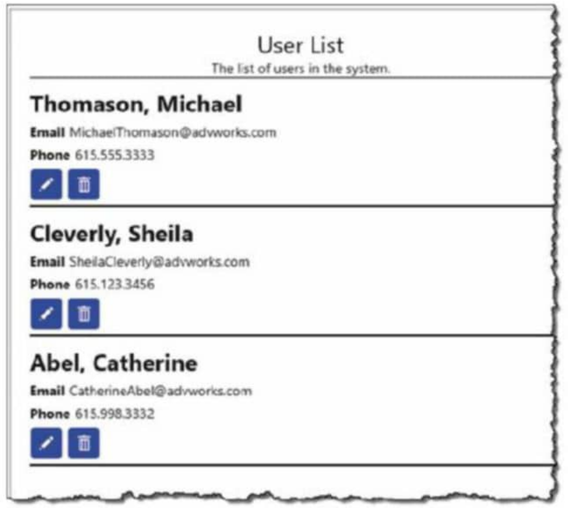
```
<Border Style="{StaticResource Border.Page}">
  <Grid Style="{StaticResource Grid.Page}">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <partial:HeaderView
      ViewTitle="User List"
      ViewDescription="The list of users
        in the system." />

    <ListView Grid.Row="1"
       SeparatorColor="Black"
       SeparatorVisibility="Default"
       ItemsSource="{Binding Users}">
      <ListView.ItemTemplate>
        <DataTemplate x:DataType="model:User">
          <TextCell Text="{Binding FullName}"
                Detail="{Binding Email}" />
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </Grid>
</Border>
```

Listing 3: Call the GetAsync() method on the _ViewModel variable to retrieve the list of users to display on this page.

```
using AdventureWorks.MAUI.MauiViewModelClasses;

namespace AdventureWorks.MAUI.Views;

public partial class UserListView
  : ContentPage {
  public UserListView(UserViewModel viewModel) {
    InitializeComponent();

    _ViewModel = viewModel;
  }

  private readonly UserViewModel _ViewModel;

  protected async override void OnAppearing() {
    base.OnAppearing();

    BindingContext = _ViewModel;

    await _ViewModel.GetAsync();
  }
}
```

Listing 4: Use a ViewCell to provide the most consistent UI between all platforms.

```
<ViewCell>
  <VerticalStackLayout Spacing="5">
    <HorizontalStackLayout>
      <Label FontAttributes="Bold"
        FontSize="Title"
        Text="{Binding LastNameFirstName}" />
    </HorizontalStackLayout>
    <HorizontalStackLayout>
      <Label FontAttributes="Bold"
             Text="Email" />
      <Label Text="{Binding Email}" />
    </HorizontalStackLayout>
    <HorizontalStackLayout>
      <Label FontAttributes="Bold"
             Text="Phone" />
      <Label Text="{Binding Phone}" />
    </HorizontalStackLayout>
    <HorizontalStackLayout>
      <ImageButton Source="edit.png"
             ToolTipProperties.Text="Edit User" />
      <ImageButton Source="trash.png"
             ToolTipProperties.Text="Delete User" />
    </HorizontalStackLayout>
    <BoxView Style="{StaticResource
      Grid.Item.Separator}" />
  </VerticalStackLayout>
</ViewCell>
```



User List

The list of users in the system.

Thomason, Michael

Email MichaelThomason@adwworks.com

Phone 615.555.3333

Cleverly, Sheila

Email SheilaCleverly@adwworks.com

Phone 615.123.3456

Abel, Catherine

Email CatherineAbel@adwworks.com

Phone 615.998.3332

Figure 3: Create your own custom template using ViewCell.

Using the ViewCell

Open the **Views\UserListView.xaml** file and replace the <TextCell> element with the XAML shown in **Listing 4**. When you use a ViewCell control, you have complete flexibility to use any layout you want for each row in your collection. In this ViewCell, create a VerticalStackLayout element with a spacing of five device-independent pixels between each child element. Four HorizontalStackLayout elements are created with Label controls in each to define where the various properties of each User object are to be displayed (see **Figure 3**). The last element within the VerticalStackLayout is a BoxView control with a keyed style attached to it.

Open the **Resources\Styles\CommonStyles.xaml** file located in the Common.Library.MAUI file and add the new **keyed** style for the BoxView control. This style is used to separate each user in the ListView control, as shown in the following XAML:

```
<Style TargetType="BoxView"
       x:Key="Grid.Item.Separator">
  <Setter Property="VerticalOptions"
          Value="Fill" />
  <Setter Property="BackgroundColor"
          Value="Black" />
  <Setter Property="HeightRequest"
          Value="2" />
  <Setter Property="Margin"
          Value="0,0,0,10" />
 </Style>
```

When you use a ViewCell template, add the HasUnevenRows="True" to the ListView control or the list won't render correctly on mobile devices. Remove the **SeparatorColor** and **SeparatorVisibility** properties and add the **HasUnevenRows="True"** attribute to the ListView control, as shown in the following code:

```
<ListView Grid.Row="1"
          HasUnevenRows="True"
          ItemsSource="{Binding Users}">
```

Try It Out

Run the application and click on the **Users** menu to see the new user template as was shown in **Figure 3**. If you run the application on the Android emulator or on iOS, it should look the same as on Windows. This is the main reason I like to use ViewCell: The results on each platform almost always look the same.

Display User Detail View from List View

Because you removed the button that read **Navigation to Detail** from the user list page, you removed the ability to navigate to the user detail page. Let's now add that capability back in by clicking on the edit button on one of the users in the list.

Add Command for Editing

Open the **MauiViewModelClasses\UserViewModel.cs** file and add a new method to navigate to the user detail page. As mentioned previously, .NET MAUI uses a URI-based navigation system. As such, you pass parameters between pages by using a key/value pair as you would for a web application. The first key/value pair is separated from the name of the page to navigate to by a question mark, whereas succeeding key/value pairs are separated using an ampersand. If you type in the following code, be sure there are no spaces within the interpolated string.

```
#region EditAsync Method
protected async Task EditAsync(int id) {
  await Shell.Current.GoToAsync(
    $"{nameof(Views.UserDetailView)}?id={id}");
}
#endregion
```

Define an **EditCommand** property of the type ICommand to which you can map the **Command** property on the Edit button, as shown in the following code snippet.

```
public ICommand? EditCommand { get; private set; }
```

Create an instance of this new command property in the Init() method and set it to call the EditAsync() method you just created.

```
EditCommand = new Command<int>(
  async (int id) => await EditAsync(id));
```

The edit button is contained within the DataTemplate in the ListView control. As such, the **x:DataType** is mapped to the **User** object and not the user view model class. Because the EditCommand property you just created is in the view model, you need a reference back to that view model. Open the **Views\UserListView.xaml** file and add an **x:Name** attribute to the ContentPage.

```
x:Name="UserListPage"
```

Now that you've defined a name that refers to the ContentPage, and that page has its **x:DataType** equal to the UserViewModel object, this allows you to reference that view model by adding a **Source={x:Reference UserListPage}** within the **Binding** markup extension. Locate the ImageButton control for editing and make it look like the following XAML.

```
<ImageButton Source="edit.png"
  ToolTipProperties.Text="Edit User"
  CommandParameter="{Binding UserId}"
  Command="{Binding
    Source={x:Reference UserListPage},
    Path=BindingContext.EditCommand}" />
```

Receive User ID in Detail Page

The UserId is passed to the user detail page as the key ID in your navigation URI. You need some way to retrieve that ID key and assign the value to a property in the UserDetailView class. Open the **Views\UserDetailView.xaml.cs** file and first add a public property named **UserId**.

```
public int UserId { get; set; }
```

Next, add a **[QueryProperty]** attribute above the **public partial class UserDetailView** definition, as shown in the code snippet below. The **QueryProperty** attribute reads the value passed as part of the key pair "id=2", for example. It maps that value to the property name specified as the first parameter to the QueryProperty attribute. In this case, it maps it to the **UserId** property you just created.

```
[QueryProperty(nameof(UserId), "id")]
public partial class UserDetailView
  : ContentPage {
   // REST OF THE CODE HERE
}
```

Locate the OnAppearing() event procedure and where you call the method GetAsync(1) with the hard-coded value one, use the **UserId** property, as shown in the code below.

```
// Retrieve a User
await _ViewModel.GetAsync(UserId);
```

Try It Out

Run the application and click on the **User** menu. **Click** on the edit button for the first users and the detail for the first user is displayed on the detail page (**Figure 4**). Notice that a back arrow is displayed in the upper left corner of the window shell. If you click on this, you're returned to the user list page. If you click on each user

dtSearch®

Instantly Search Terabytes

Enterprise and developer products have:
- over 25 different search features
- credit card search and other forensics-oriented options
- efficient multithreaded indexing and searching

dtSearch's document filters support:
- local and remote "Office" files, PDFs, compression formats, etc.
- emails with multilevel attachments
- a wide variety of databases
- web data

Developers:
- SDKs for Win / Linux / macOS
- APIs for current .NET / C++ / Java
- faceted search, granular data classification and other API options
- deploy on-premises or in cloud

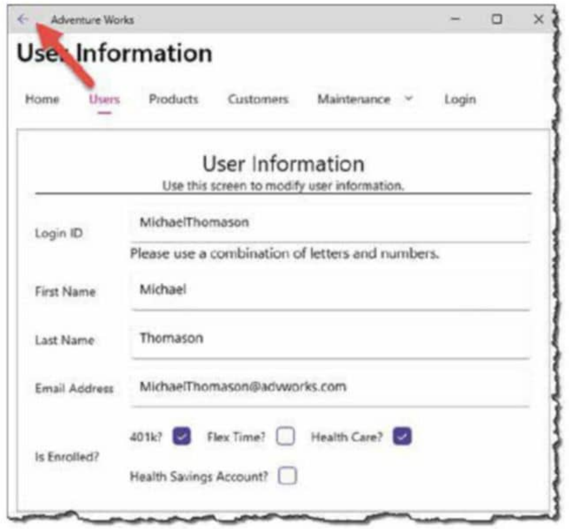
Visit dtSearch.com
- for hundreds of reviews and case studies
- for fully-functional enterprise and developer evaluations

The Smart Choice for Text Retrieval® since 1991

dtSearch.com | 1-800-IT-FINDS

Getting the Sample Code

You can download the sample code for this article by visiting www.CODEMag.com under the issue and article, or by visiting www.pdsa.com/downloads. Select "Articles" from the Category drop-down. Then select "Exploring .NET MAUI: Working with Lists of Data" from the Item drop-down.



User Information

Use this screen to modify user information.

Login ID: MichaelThomason

Please use a combination of letters and numbers.

First Name: Michael

Last Name: Thomason

Email Address: MichaelThomason@adworks.com

Is Enrolled? 401k? ☑ Flex Time? ☐ Health Care? ☑ Health Savings Account? ☐

Figure 4: After navigating to the detail page, a back arrow is displayed to allow you to go back to the list page.

in the list, you're directed to the detail page and the **UserId** value for each user is passed to the detail page. The code behind on the detail page runs and the **UserId** value is passed to the GetAsync(id) method, which in turn fills the **CurrentEntity** property on the view model. Each control on the detail page is bound to a property on the **CurrentEntity** property, thus the data for the current user is displayed on the detail page.

Save User and Return to User List View

When the user clicks on the Save button on the detail page, a few different things may happen. The data is saved, and they are returned to the list page to see their changes appear in the list. If there are one or more validation errors that occur due to bad input, a list of values to update is displayed. Another scenario is that an exception happens and an error message is displayed. You have not written code to save the data to a data store yet, but let's at least take the happy path and assume the data is saved, and you want to return the user to the list page. Open the **MauiViewModelClasses\UserViewModel.cs** file and add a **SaveAsync()** method, as shown in the following code:

```
#region SaveAsync Method
public override async Task<User?> SaveAsync() {
  User? ret = await base.SaveAsync();

  if (ret != null) {
    await Shell.Current.GoToAsync("..");
  }

  return ret;
}
#endregion
```

This method overrides the SaveAsync() in the UserViewModel base class. It calls the SaveAsync() method in the base class, which, if the user is saved successfully, a valid User object is returned. If the User object is valid, then use the GoToAsync() method to navigate back to the list page. The two dots in the double quotes tell the .NET MAUI navigation system to go back one level, or to the page that called this one. Open the **ViewModelClasses\UserViewModel.cs** file and **remove** the **System.Diagnostics.Debugger.Break();** line of code from the SaveAsync() method so you can see this code navigate back to the list page after clicking on the Save button.

Try It Out

Run the application and click on the **User** menu. Click on the edit button for a user to see the user detail page appear. Click on the **Save** button and you're redirected back to the user list page. There's no code to save any changes yet, but this will come in a later article.

Cancel Changes and Return to User List View

Besides the Save button, there is also a Cancel button on the user detail page. For the Cancel button, you want to just have it redirect back to the user list page without saving any changes made on the page. Open the **MauiViewModelClasses\UserViewModel.cs** file and add a **CancelAsync()** method. This method simply navigates back to the user list view using the GoToAsync() method.

```
#region CancelAsync Method
public async Task CancelAsync() {
  await Shell.Current.GoToAsync("..");
}
#endregion
```

As you've done previously, add a **CancelCommand** property to which you can map the **Command** property to the Cancel button.

```
public ICommand? CancelCommand
  { get; private set; }
```

Create an instance of this new command property in the Init() method and set it to call the CancelAsync() method you just created.

```
CancelCommand = new Command(
  async () => await CancelAsync());
```

Open the **Views\UserDetailView.xaml** file, locate the Cancel button and add a Command attribute that's bound to the **CancelCommand** property.

```
<Button Text="Cancel"
        ImageSource="cancel.png"
        ContentLayout="Left"
        ToolTipProperties.Text="Cancel Changes"
        Command="{Binding CancelCommand}" />
```

Try It Out

Run the application and click on the **User** menu. Click on the **Edit** button for a user to see the detail page appear. Click on the **Cancel** button and you're redirected back to the user list page.

Display a List of Products Using a CollectionView

You've seen the ListView control. Let's now build the product list page and product detail page using a **CollectionView** control. The **CollectionView** control is like the ListView in that it's used to present a vertical list of data to the user. The CollectionView control doesn't have any Cell controls. Instead, you create the DataTemplate in any format you wish. This provides the same flexibility as using the ViewCell you used on the ListView control. With the CollectionView control, you may select single or multiple items. The CollectionView control also uses virtualization, thus making it potentially more performant than the ListView.

To illustrate the CollectionView control, let's start building the product classes to display a list of products and display a single product object. Create a product entity class to represent each field of a product. Create a Repository class to return a list of product data, as well as a single product. Create a product view model class to support displaying a list of products and selecting a single product, just like you did with the user view model class.

Create a Product Entity Class

Right mouse-click on the **EntityClasses** in the **AdventureWorks.EntityLayer** project and add a new class named **Product**. Replace the entire contents of this new file with the code shown in **Listing 5**. This class is de-

Listing 5: Create a Product entity class to represent a single product.

```
using Common.Library;

namespace AdventureWorks.EntityLayer;

public partial class Product : EntityBase {
  #region Private Variables
  private int _ProductID;
  private string _Name = string.Empty;
  private string _ProductNumber
    = string.Empty;
  private string _Color = string.Empty;
  private decimal _StandardCost = 1;
  private decimal _ListPrice = 2;
  private string? _Size = string.Empty;
  private decimal? _Weight;
  private int? _ProductCategoryID;
  private int? _ProductModelID;
  private DateTime _SellStartDate;
  private DateTime? _SellEndDate;
  private DateTime? _DiscontinuedDate;
  private DateTime _ModifiedDate;
  #endregion

  #region Public Properties
  public int ProductID {
    get { return _ProductID; }
    set {
      _ProductID = value;
      RaisePropertyChanged(nameof(ProductID));
    }
  }

  public string Name {
    get { return _Name; }
    set {
      _Name = value;
      RaisePropertyChanged(nameof(Name));
    }
  }

  public string ProductNumber {
    get { return _ProductNumber; }
    set {
      _ProductNumber = value;
      RaisePropertyChanged(nameof(ProductNumber));
    }
  }

  public string Color {
    get { return _Color; }
    set {
      _Color = value;
      RaisePropertyChanged(nameof(Color));
    }
  }

  public decimal StandardCost {
    get { return _StandardCost; }
    set {
      _StandardCost = value;
      RaisePropertyChanged(nameof(StandardCost));
    }
  }

  public decimal ListPrice {
    get { return _ListPrice; }
    set {
      _ListPrice = value;
      RaisePropertyChanged(nameof(ListPrice));
    }
  }

  public string? Size {
    get { return _Size; }
    set {
      _Size = value;
      RaisePropertyChanged(nameof(Size));
    }
  }

  public decimal? Weight {
    get { return _Weight; }
    set {
      _Weight = value;
      RaisePropertyChanged(nameof(Weight));
    }
  }

  public int? ProductCategoryID {
    get { return _ProductCategoryID; }
    set {
      _ProductCategoryID = value;
      RaisePropertyChanged(nameof(ProductCategoryID));
    }
  }

  public int? ProductModelID {
    get { return _ProductModelID; }
    set {
      _ProductModelID = value;
      RaisePropertyChanged(nameof(ProductModelID));
    }
  }

  public DateTime SellStartDate {
    get { return _SellStartDate; }
    set {
      _SellStartDate = value;
      RaisePropertyChanged(nameof(SellStartDate));
    }
  }

  public DateTime? SellEndDate {
    get { return _SellEndDate; }
    set {
      _SellEndDate = value;
      RaisePropertyChanged(nameof(SellEndDate));
    }
  }

  public DateTime? DiscontinuedDate {
    get { return _DiscontinuedDate; }
    set {
      _DiscontinuedDate = value;
      RaisePropertyChanged(nameof(DiscontinuedDate));
    }
  }

  public DateTime ModifiedDate {
    get { return _ModifiedDate; }
    set {
      _ModifiedDate = value;
      RaisePropertyChanged(nameof(ModifiedDate));
    }
  }
  #endregion
}
```

signed exactly like the User class where it has private variables mapped to public properties, and which raise the PropertyChanged event when modified.

Create a Product View Model Class

Right mouse-click on the **ViewModelClasses** in the **AdventureWorks.ViewModelLayer** project and add a new class named **ProductViewModel**. Replace the entire contents of this new file with the code shown in **Listing 6**. This view model class uses the same design pattern as the UserViewModel class you created earlier. It has properties for a repository object, a list of products, and the currently selected product. It also contains the same three methods as in the UserViewModel class: GetAsync(), GetAsync(id), and SaveAsync().

Create a Product View Model for Commanding

Just like you did with the UserViewModel class, create a view model class in the .NET MAUI application to handle commanding for calling methods in the view model. In the **AdventureWorks.MAUI** project, right mouse-click on the **MauiViewModelClasses** folder and create a new class named **ProductViewModel**. Replace the entire contents of this new file with the code shown in **Listing 7**. You'll add more to this class a little later.

Create a Product Repository Class

Right mouse-click on the **RepositoryClasses** in the **AdventureWorks.DataLayer.Mock** project and add a new class named **ProductRepository**. Replace the entire contents of this new file with the code shown in **Listing 8**.

Listing 6: Create a Product view model for data binding on the product pages.

```
using AdventureWorks.EntityLayer;
using Common.Library;
using System.Collections.ObjectModel;

namespace AdventureWorks.ViewModelLayer;

public class ProductViewModel : ViewModelBase {
  #region Constructors
  public ProductViewModel() : base() {
  }

  public ProductViewModel(
    IRepository<Product> repo) : base() {
    _Repository = repo;
  }
  #endregion

  #region Private Variables
  private readonly IRepository<Product>? _Repository;

  private ObservableCollection<Product> _Products = new();
  private Product? _CurrentEntity = new();
  #endregion

  #region Public Properties
  public ObservableCollection<Product> Products {
    get { return _Products; }
    set {
      _Products = value;
      RaisePropertyChanged(nameof(Products));
    }
  }

  public Product? CurrentEntity {
    get { return _CurrentEntity; }
    set {
      _CurrentEntity = value;
      RaisePropertyChanged(nameof(CurrentEntity));
    }
  }
  #endregion

  #region GetAsync Method
  public async
    Task<ObservableCollection<Product>>
      GetAsync() {
    RowsAffected = 0;

    try {
      if (_Repository == null) {
        LastErrorMessage = REPO_NOT_SET;
      }
      else {
        Products = await _Repository.GetAsync();
        RowsAffected = Products.Count;
        InfoMessage = $"Found {RowsAffected} Products";
      }
    }
    catch (Exception ex) {
      PublishException(ex);
    }

    return Products;
  }
  #endregion

  #region GetAsync(id) Method
  public async Task<Product?>
    GetAsync(int id) {
    try {
      // Get a Product from a data store
      if (_Repository != null) {
        CurrentEntity = await _Repository.GetAsync(id);
        if (CurrentEntity == null) {
          InfoMessage = $"Product id={id} was not found.";
        }
        else {
          InfoMessage = "Found the Product";
        }
      }
      else {
        LastErrorMessage = REPO_NOT_SET;
        InfoMessage = "Found a MOCK Product";

        // MOCK Data
        CurrentEntity =
          await Task.FromResult(new Product {
          ProductID = id,
          Name = "A New Product",
          Color = "Black",
          StandardCost = 10,
          ListPrice = 20,
          SellStartDate = Convert.ToDateTime("7/1/2023"),
          Size = "LG"
        });
      }

      RowsAffected = 1;
    }
    catch (Exception ex) {
      RowsAffected = 0;
      PublishException(ex);
    }

    return CurrentEntity;
  }
  #endregion

  #region SaveAsync Method
  public async virtual Task<Product?>
    SaveAsync() {
    // TODO: Write code to save data

    return await Task.FromResult(new Product());
  }
  #endregion
}
```

The code in the ProductRepository follows the same design pattern you already learned building the UserRepository class. The GetAsync() method returns a set of four product objects and the GetAsync(id) returns a single product object.

Build a Product List Page

You already have a product detail page, so now you need to create a page to display a list of products. Right mouse-click on the **Views** folder and add a new **Content Page (XAML)** named **ProductListView**. Change the **Title** attribute to "Product List". Add three XML namespaces to reference the partial views, the MAUI view models, and the entity layer project.

```
xmlns:partial="clr-namespace:
  AdventureWorks.MAUI.ViewsPartial"
xmlns:vm="clr-namespace:
  AdventureWorks.MAUI.MauiViewModelClasses"
```

```
xmlns:model="clr-namespace:
  AdventureWorks.EntityLayer;
    assembly=AdventureWorks.EntityLayer"
```

Listing 7: Create a view model in the .NET MAUI application for working with product data.

```
using AdventureWorks.EntityLayer;
using Common.Library;

namespace AdventureWorks.MAUI.MauiViewModelClasses;

public class ProductViewModel :
  AdventureWorks.ViewModelLayer.ProductViewModel {
  #region Constructors
  public ProductViewModel() : base() {
  }

  public ProductViewModel(
    IRepository<Product> repo) : base(repo) {
  }
  #endregion
}
```

Listing 8: Create a product repository class to return a set of product objects.

```
using AdventureWorks.EntityLayer;
using Common.Library;
using System.Collections.ObjectModel;

namespace AdventureWorks.DataLayer;

/// <summary>
/// Creates a set of Product mock data
/// </summary>
public partial class ProductRepository
  : IRepository<Product> {
  #region GetAsync Method
  public async
    Task<ObservableCollection<Product>>
      GetAsync() {
    return await Task.FromResult(
      new ObservableCollection<Product>
    {
      new() {
        ProductID = 680,
        Name = @"HL Road Frame - Black, 58",
        ProductNumber = @"FR-R92B-58",
        Color = @"Black",
        StandardCost = 100.0000m,
        ListPrice = 1431.5000m,
        Size = @"58",
        Weight = 1016.04m,
        ProductCategoryID = 18,
        ProductModelID = 6,
        SellStartDate
          = new DateTime(2002, 6, 1),
        SellEndDate = null,
        DiscontinuedDate = null,
        ModifiedDate
          = new DateTime(2008, 3, 11),
      },
      new() {
        ProductID = 707,
        Name = @"Sport-100 Helmet, Red",
        ProductNumber = @"HL-U509-R",
        Color = @"Red",
        StandardCost = 13.0863m,
        ListPrice = 34.9900m,
        Size = null,
        Weight = 3.4m,
        ProductCategoryID = 35,
        ProductModelID = 33,
        SellStartDate
          = new DateTime(2005, 7, 1),
        SellEndDate = null,
        DiscontinuedDate = null,
        ModifiedDate
          = new DateTime(2008, 3, 11),
      },
      new() {
        ProductID = 712,
        Name = @"AWC Logo Cap",
        ProductNumber = @"CA-1098",
        Color = @"Multi",
        StandardCost = 6.9223m,
        ListPrice = 8.9900m,
        Size = null,
        Weight = 0.80m,
        ProductCategoryID = 23,
        ProductModelID = 2,
        SellStartDate
          = new DateTime(2005, 7, 1),
        SellEndDate = null,
        DiscontinuedDate = null,
        ModifiedDate
          = new DateTime(2008, 3, 11),
      },
      new() {
        ProductID = 713,
        Name = @"Long-Sleeve Logo Jersey, S",
        ProductNumber = @"LJ-0192-S",
        Color = @"Multi",
        StandardCost = 38.4923m,
        ListPrice = 49.9900m,
        Size = "S",
        Weight = null,
        ProductCategoryID = 25,
        ProductModelID = 11,
        SellStartDate
          = new DateTime(2005, 7, 1),
        SellEndDate = null,
        DiscontinuedDate = null,
        ModifiedDate
          = new DateTime(2008, 3, 11),
      }
    });
  }
  #endregion

  #region GetAsync(id) Method
  public async Task<Product?> GetAsync(int id) {
    ObservableCollection<Product> list
      = await GetAsync();
    Product? entity = list.Where(
      row => row.ProductID == id)
      .FirstOrDefault();

    return entity;
  }
  #endregion
}
```

Add a **x:DataType** attribute to the <ContentPage> element to take advantage of compiled bindings on the page.

```
x:DataType="vm:ProductViewModel"
```

Listing 9: Create the product list page using a CollectionView control.

```
<Border Style="{StaticResource Border.Page}">
  <Grid Style="{StaticResource Grid.Page}">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <partial:HeaderView ViewTitle="Product List"
      ViewDescription="The list of products in
        the system." />

    <CollectionView Grid.Row="1"
                    SelectionMode="Single"
                    ItemsSource="{Binding Products}">
      <CollectionView.ItemTemplate>
        <DataTemplate x:DataType="model:Product">
          <VerticalStackLayout Spacing="5">
            <HorizontalStackLayout>
              <Label FontAttributes="Bold"
                     FontSize="Title"
                     Text="{Binding Name}" />
            </HorizontalStackLayout>
            <HorizontalStackLayout>
              <Label FontAttributes="Bold"
                     Text="Color" />
              <Label Text="{Binding Color}" />
            </HorizontalStackLayout>
            <HorizontalStackLayout>
              <Label FontAttributes="Bold"
                     Text="Price" />
              <Label Text="{Binding ListPrice,
                     StringFormat='{0:c}'}" />
            </HorizontalStackLayout>
            <HorizontalStackLayout>
              <ImageButton Source="edit.png"
                ToolTipProperties.Text="Edit Product" />
              <ImageButton Source="trash.png"
                ToolTipProperties.Text="Delete Product" />
            </HorizontalStackLayout>
            <BoxView Style="{StaticResource Grid.Item.Separator}" />
          </VerticalStackLayout>
        </DataTemplate>
      </CollectionView.ItemTemplate>
    </CollectionView>
  </Grid>
</Border>
```

Listing 10: Add code to get all products for displaying on the product list page.

```
using AdventureWorks.MAUI.MauiViewModelClasses;

namespace AdventureWorks.MAUI.Views;

public partial class ProductListView
  : ContentPage {
  public ProductListView(ProductViewModel viewModel) {
    InitializeComponent();

    _ViewModel = viewModel;
  }

  private readonly ProductViewModel _ViewModel;

  protected async override void OnAppearing() {
    base.OnAppearing();

    BindingContext = _ViewModel;

    await _ViewModel.GetAsync();
  }
}
```

Replace the <VerticalStackLayout> with the XAML shown in **Listing 9**. Instead of a ListView control, use a CollectionView control. Specify that you only want to allow a single row to be selected at a time by setting the **SelectionMode** property to **Single**. Bind the **ItemsSource** property to the Products collection in the ProductViewModel class. The <CollectionView.ItemTemplate> element specifies the DataTemplate for how to display each item in your collection. Always add the x:DataType attribute to the DataTemplate so it knows what kind of entity object is within each row of your collection.

Why is there an ItemTemplate element around the DataTemplate? It's because there is also a HeaderTemplate element and a FooterTemplate element that you may add to the CollectionView. These templates may also have a DataTemplate that describes how the header and footer are laid out for the CollectionView. I don't generally use these templates, but you may find them handy in some circumstances. Open the **Views\ProductListView.xaml.cs** file and replace the entire contents of this file with the code shown in **Listing 10**. Once again, this code should look very familiar to you, as it's almost identical to the code in the User List View code behind.

Change Application Shell and DI for Products

Now that you added a product list page and modified both the list and the product detail page to have the ProductViewModel injected into each page, you need to ensure that these pages work with DI. Open the **AppShell.xaml** file and change the <ShellContent> element for the Products tab to point to the **ProductListView** page instead of the **ProductDetailView** page, as shown in the following XAML:

```
<ShellContent Title="Products"
  ContentTemplate="{DataTemplate
    views:ProductListView}" />
```

Open the **ExtensionClasses\ServiceExtensions.cs** file and, in the AddRepositoryClasses() method, add code to inject the ProductRepository class.

```
services.AddScoped<IRepository<Product>,
  ProductRepository>();
```

In the AddViewModelClasses() method, add code to inject the ProductViewModel.

```
services.AddScoped<MauiViewModelClasses
  .ProductViewModel>();
```

In the AddViewClasses() method, add code to inject both the product detail and list pages.

```
services.AddScoped<ProductDetailView>();
services.AddScoped<ProductListView>();
```

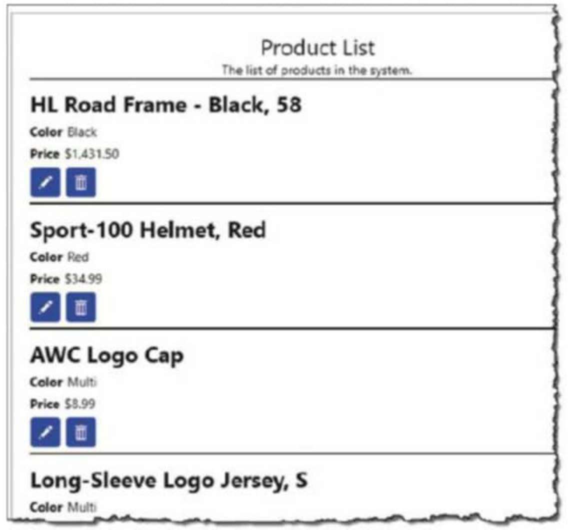
Try It Out

Run the application and click on the **Products** menu to see the list of products, as shown in **Figure 5**.

Display Product Detail from Collection View

Let's now modify the pages so the list can navigate to the product detail page when the edit button is clicked. Open the **Views\ProductDetailView.xaml** file and add an XML namespace to the product view model class in the MauiViewModelClasses namespace.

```
xmlns:vm="clr-namespace:
  AdventureWorks.MAUI.MauiViewModelClasses"
```



Product List
The list of products in the system.

HL Road Frame - Black, 58
Color Black
Price $1,431.50

Sport-100 Helmet, Red
Color Red
Price $34.99

AWC Logo Cap
Color Multi
Price $8.99

Long-Sleeve Logo Jersey, S
Color Multi

Figure 5: A product list is created using a CollectionView control.

Add the **x:DataType** on the ContentPage element to specify that the ProductViewModel class is the source of the data binding for this page.

```
x:DataType="vm:ProductViewModel"
```

The XAML shown in **Listing 11** is all the controls on the ProductDetailView page with their data bindings to the appropriate properties to the Product class. This XAML isn't too different from what you created in an earlier article; it just has the data bindings attached to the controls.

Replace the Product Detail View Code Behind

Open the **Views\ProductDetailView.xaml.cs** file and replace the entire contents of this file with the code shown

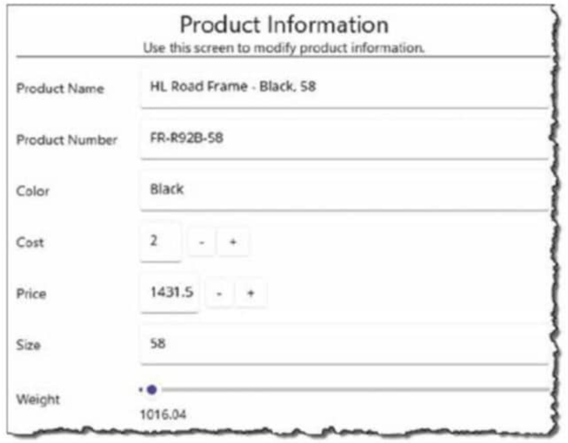
Listing 11: Map the data entry controls on the product detail page to the appropriate properties in the Product class.

```
<Label Text="Product Name"
       Grid.Row="1" />
<Entry Grid.Column="1"
       Grid.Row="1"
       Text="{Binding CurrentEntity.Name}" />
<Label Text="Product Number"
       Grid.Row="2" />
<Entry Grid.Row="2"
       Grid.Column="1"
       Text="{Binding CurrentEntity.ProductNumber}" />
<Label Text="Color"
       Grid.Row="3" />
<Entry Grid.Row="3"
       Grid.Column="1"
       Text="{Binding CurrentEntity.Color}" />
<Label Text="Cost"
       Grid.Row="4" />
<HorizontalStackLayout Grid.Row="4"
                       Grid.Column="1">
  <Entry Text="{Binding CurrentEntity.StandardCost}" />
  <Stepper Value="{Binding CurrentEntity.StandardCost}"
           Minimum="1"
           Maximum="{Binding CurrentEntity.ListPrice}"
           Increment="1" />
</HorizontalStackLayout>
<Label Text="Price"
       Grid.Row="5" />
<HorizontalStackLayout Grid.Row="5"
                       Grid.Column="1">
  <Entry Text="{Binding CurrentEntity.ListPrice}" />
  <Stepper Value="{Binding CurrentEntity.ListPrice}"
           Minimum="{Binding CurrentEntity.StandardCost}"
           Maximum="9999"
           Increment="1" />
</HorizontalStackLayout>
<Label Text="Size"
       Grid.Row="6" />
<Entry Grid.Row="6"
       Grid.Column="1"
       Text="{Binding CurrentEntity.Size}" />
<Label Text="Weight"
       Grid.Row="7" />
<VerticalStackLayout Grid.Row="7"
                     Grid.Column="1">
  <Slider x:Name="weight"
          Value="{Binding CurrentEntity.Weight}"
          Minimum="1"
          Maximum="99999" />
  <Label Text="{Binding CurrentEntity.Weight}" />
</VerticalStackLayout>
```

```
<Label Text="Category"
       Grid.Row="8" />
<Entry Grid.Row="8"
       Grid.Column="1"
       Text="{Binding CurrentEntity.ProductCategoryID}" />
<Label Text="Model"
       Grid.Row="9" />
<Entry Grid.Row="9"
       Grid.Column="1"
       Text="{Binding CurrentEntity.ProductModelID}" />
<Label Text="Selling Start Date"
       Grid.Row="10" />
<DatePicker Grid.Row="10"
            Grid.Column="1"
            Date="{Binding CurrentEntity.SellStartDate}" />
<Label Text="Selling End Date"
       Grid.Row="11" />
<DatePicker Grid.Row="11"
            Grid.Column="1"
            Date="{Binding CurrentEntity.SellEndDate}" />
<Label Text="Discontinued Date"
       Grid.Row="12" />
<DatePicker Grid.Row="12"
            Grid.Column="1"
            Date="{Binding CurrentEntity.DiscontinuedDate}" />
<Label Text="Product Picture"
       Grid.Row="13" />
<Image Grid.Row="13"
       Grid.Column="1"
       HorizontalOptions="Start"
       Aspect="Center"
       Source="bikeframe.jpg" />
<Label Text="Product Notes"
       Grid.Row="14" />
<Editor Grid.Row="14"
        Grid.Column="1"
        HeightRequest="100" />
<HorizontalStackLayout Grid.Row="15"
                       Grid.Column="1">
  <Button Text="Save"
    ImageSource="save.png"
    ContentLayout="Left"
    ToolTipProperties.Text="Save Data"
    Command="{Binding SaveCommand}" />
  <Button Text="Cancel"
    ImageSource="cancel.png"
    ContentLayout="Left"
    ToolTipProperties.Text="Cancel Changes"
    Command="{Binding CancelCommand}" />
</HorizontalStackLayout>
```

Listing 12: Add code in the product detail page to retrieve a single product object.

```
using AdventureWorks.MAUI.MauiViewModelClasses;

namespace AdventureWorks.MAUI.Views;

[QueryProperty(nameof(ProductId), "id")]
public partial class ProductDetailView
  : ContentPage {
  public ProductDetailView(
    ProductViewModel viewModel) {
    InitializeComponent();

    _ViewModel = viewModel;
  }

  private readonly ProductViewModel _ViewModel;
  public int ProductId { get; set; }

  protected async override void OnAppearing() {
    base.OnAppearing();

    // Set the Page BindingContext
    BindingContext = _ViewModel;

    // Retrieve a Product
    await _ViewModel.GetAsync(ProductId);
  }
}
```


Product Information

Use this screen to modify product information.

Product Name: HL Road Frame - Black, 58

Product Number: FR-R92B-58

Color: Black

Cost: 2

Price: 1431.5

Size: 58

Weight: 1016.04

Figure 6: Display the product detail information by navigating from a product on the list page.

in **Listing 12**. The product detail page must accept a ProductId passed from the product list page, so add a [QueryProperty] attribute to the ProductDetailView class. Add the OnAppearing() method to set the **BindingContext** of the page to the view model injected into the constructor. Call the GetAsync(ProductId) method to load the **CurrentEntity** property with the product data to display on the controls.

Modify Product View Model Commands

Open the **MauiViewModelClasses\ProductViewModel.cs** file you created earlier and add the code shown in **Listing 13** after the constructors. This code should look very familiar, as you used almost the exact same code in the UserViewModel that handles the commands for the user pages.

Modify Product List View XAML

Add the code to the edit button on the product list page to bind to the **EditCommand** property in the ProductViewModel class. Open the **Views\ProductViewList.xaml** file and add an **x:Name** attribute to the ContentPage starting tag, as shown in the following XAML:

```
x:Name="ProductListPage"
```

Now that you've defined a name that refers to the ContentPage, and that page has as its x:DataType equal to the ProductViewModel object, this allows you to reference that view model by adding a **Source={x:Reference ProductListPage}** within the Binding markup extension. Locate the ImageButton control for the edit button and make it look like the following XAML:

```
<ImageButton Source="edit.png"
  ToolTipProperties.Text="Edit Product"
  CommandParameter="{Binding ProductID}"
  Command="{Binding
    Source={x:Reference ProductListPage},
    Path=BindingContext.EditCommand}" />
```

Because you changed the <ShellContent> element to call the ProductListView instead of the ProductDetailView page, that page is no longer in the list of routes. Open the **AppShell.xaml.cs** file and register the route for the ProductDetailView in the constructor, as shown in the following code:

```
Routing.RegisterRoute(
  nameof(Views.ProductDetailView),
  typeof(Views.ProductDetailView));
```

Listing 13: Add the commands to handle editing, saving, and cancelling on the product pages.

```
#region Commands
public ICommand? SaveCommand { get; private set; }
public ICommand? CancelCommand { get; private set; }
public ICommand? EditCommand { get; private set; }
#endregion

#region Init Method
public override void Init() {
  base.Init();

  // Create commands for this view
  SaveCommand = new Command(
    async () => await SaveAsync());
  CancelCommand = new Command(
    async () => await CancelAsync());
  EditCommand = new Command<int>(
    async (int id) => await EditAsync(id));
}
#endregion

#region SaveAsync Method
protected new async Task<Product?> SaveAsync() {
  Product? ret = await base.SaveAsync();

  if (ret != null) {
    await Shell.Current.GoToAsync("..");
  }

  return ret;
}
#endregion

#region CancelAsync Method
protected async Task CancelAsync() {
  await Shell.Current.GoToAsync("..");
}
#endregion

#region EditAsync Method
protected async Task EditAsync(int id) {
  await Shell.Current.GoToAsync(
  $"{nameof(Views.ProductDetailView)}?id={id}");
}
#endregion
```

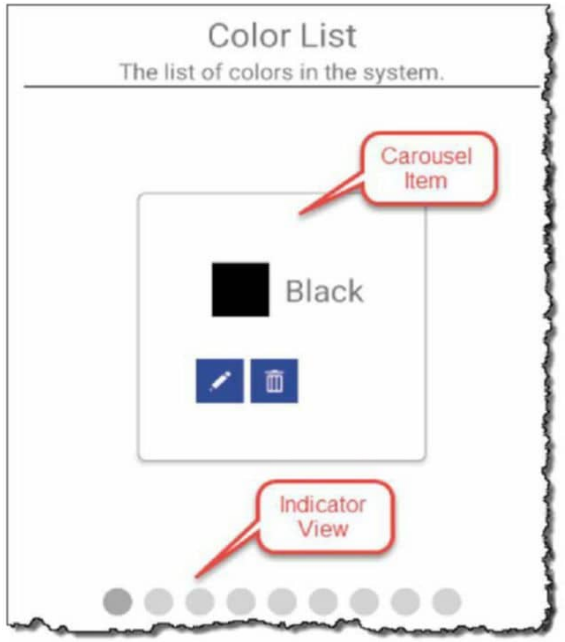


Figure 7: Use the CarouselView and IndicatorView controls together.

Try It Out

Run the application and click on the **Product** menu. Click on the edit button for different products to see the product detail page appear as shown in **Figure 6**. Click on the **Save** or **Cancel** buttons and you are redirected back to the product list page.

Display Colors Using a Carousel View (Mobile Development Only)

By default, a CarouselView control displays items horizontally so you can swipe back and forth to navigate from one item to another. Swiping backward from the first item

Listing 14: Create an entity class to represent a color.

```
using Common.Library;

namespace AdventureWorks.EntityLayer;

public class Color : EntityBase {
  #region Private Variables
  private int _ColorId = 0;
  private string _ColorName = string.Empty;
  #endregion

  #region Public Properties
  public int ColorId {
    get { return _ColorId; }
    set {
      _ColorId = value;
      RaisePropertyChanged(nameof(ColorId));
    }
  }

  public string ColorName {
    get { return _ColorName; }
    set {
      _ColorName = value;
      RaisePropertyChanged(nameof(ColorName));
    }
  }
  #endregion
}
```

Listing 15: Create a repository class to return a set of colors.

```
using AdventureWorks.EntityLayer;
using Common.Library;
using System.Collections.ObjectModel;

namespace AdventureWorks.DataLayer;

/// <summary>
/// Creates fake data for Colors.
/// </summary>
public partial class ColorRepository
  : IRepository<Color> {
  #region GetAsync Method
  public async
    Task<ObservableCollection<Color>>
      GetAsync() {
    return await Task.FromResult(
      new ObservableCollection<Color>
    {
      new() {
        ColorId = 1,
        ColorName = "Black",
      },
      new() {
        ColorId = 2,
        ColorName = "Blue",
      },
      new() {
        ColorId = 3,
        ColorName = "Gray",
      },
      new() {
        ColorId = 4,
        ColorName = "Multi",
      },
      new() {
        ColorId = 5,
        ColorName = "Red",
      },
      new() {
        ColorId = 6,
        ColorName = "Silver",
      },
      new() {
        ColorId = 7,
        ColorName = "Silver/Black",
      },
      new() {
        ColorId = 8,
        ColorName = "White",
      },
      new() {
        ColorId = 9,
        ColorName = "Yellow",
      }
    });
  }
  #endregion

  #region GetAsync(id) Method
  public async Task<Color?> GetAsync(int id) {
    ObservableCollection<Color> list
      = await GetAsync();
    Color? entity = list.Where(
      row => row.ColorId == id)
        .FirstOrDefault();

    return entity;
  }
  #endregion
}
```

Listing 16: Create a Color View Model class for working with the color data store.

```
using AdventureWorks.EntityLayer;
using Common.Library;
using System.Collections.ObjectModel;

namespace AdventureWorks.ViewModelLayer;

public class ColorViewModel
  : ViewModelBase {
  #region Constructors
  public ColorViewModel() : base() {
  }

  public ColorViewModel(
    IRepository<Color> repo) : base() {
    _Repository = repo;
  }
  #endregion

  #region Private Variables
  private readonly IRepository<Color>?
    _Repository;
  private ObservableCollection<Color>
    _Colors = new();
  private Color? _CurrentEntity = new();
  #endregion

  #region Public Properties
  public ObservableCollection<Color> Colors {
    get { return _Colors; }
    set {
      _Colors = value;
      RaisePropertyChanged(nameof(Colors));
    }
  }

  public Color? CurrentEntity {
    get { return _CurrentEntity; }
    set {
      _CurrentEntity = value;
      RaisePropertyChanged(nameof(CurrentEntity));
    }
  }
  #endregion

  #region GetAsync Method
  public async
    Task<ObservableCollection<Color>>
    GetAsync() {
    RowsAffected = 0;

    try {
      if (_Repository == null) {
        LastErrorMessage = REPO_NOT_SET;
      }
      else {
        Colors = await _Repository.GetAsync();
        RowsAffected = Colors.Count;
        InfoMessage =
          $"Found {RowsAffected} Colors";
      }
    }
    catch (Exception ex) {
      PublishException(ex);
    }

    return Colors;
  }
  #endregion

  #region GetAsync(id) Method
  public async Task<Color?> GetAsync(int id) {
    try {
      // Get a Color from a data store
      if (_Repository != null) {
        CurrentEntity = await _Repository.GetAsync(id);
        if (CurrentEntity == null) {
          InfoMessage = $"Color id={id} was not found.";
        }
        else {
          InfoMessage = "Found the Color";
        }
      }
      else {
        LastErrorMessage = REPO_NOT_SET;
        InfoMessage = "Found a MOCK Color";

        // MOCK Data
        CurrentEntity = await Task.FromResult(new Color {
          ColorId = 1,
          ColorName = "Black",
        });
      }

      RowsAffected = 1;
    }
    catch (Exception ex) {
      PublishException(ex);
    }
    return CurrentEntity;
  }
  #endregion

  #region SaveAsync Method
  public async virtual Task<Color?> SaveAsync() {
    // TODO: Write code to save data

    return await Task.FromResult(new Color());
  }
  #endregion
}
```

in the collection displays the last item in the collection. Similarly, swiping forward from the last item in the collection returns to the first item in the collection. Note that the CarouselView does not currently work very well in Windows applications.

CarouselView shares much of its implementation with CollectionView. However, the two controls have different use cases. CollectionView is used to present lists of data of any length, whereas CarouselView is used to display information from a small list of items.

The CarouselView works closely with an IndicatorView control (see **Figure 7**). The IndicatorView control shows how many items are in the CarouselView by using a circular dot (by default). One dot represents a single item in your collection being displayed in the CarouselView control. You may swipe left or right on the CarouselView to display each item in the collection, or you may click on one of the dots in the IndicatorView to move through the collection.

Create a Color Entity Class

When you're using a color on the product detail page, you only need to set the color name. However, in the Color entity class you're going to create, add a **ColorId** property as well, in case you wish to store colors for your products in a data store that requires a unique primary key. Right mouse-click on the **EntityClasses** folder in the **AdventureWorks.EntityLayer** project and add a new class named **Color**. Replace the entire contents of this new file with the code shown in **Listing 14**.

Create a Color Repository Class

Create a mock repository class to return a list of hard-coded Color objects. Right mouse-click on the **RepositoryClasses** folder in the **AdventureWorks.DataLayer.Mock** project and add a new class named **ColorRepository**. Replace the entire contents of this new file with the code shown in **Listing 15**. This code follows the same design pattern as that you used for the user and product repository classes.

Listing 17: Create the color view model class for commanding.

```
using Common.Library;
using System.Windows.Input;

namespace AdventureWorks.MAUI.MauiViewModelClasses;

public class ColorViewModel :
  AdventureWorks.ViewModelLayer.ColorViewModel
{
  #region Constructors
  public ColorViewModel() : base()
  {
  }

  public ColorViewModel(
    IRepository<EntityLayer.Color> repo) : base(repo)
  {
  }
  #endregion

  #region Commands
  public ICommand? SaveCommand { get; private set; }
  public ICommand? CancelCommand { get; private set; }
  public ICommand? EditCommand { get; private set; }
  #endregion

  #region Init Method
  public override void Init()
  {
    base.Init();

    // Create commands for this view
    SaveCommand = new Command(
      async () => await SaveAsync());
    EditCommand = new Command<int>(
      async (int id) => await EditAsync(id));
    CancelCommand = new Command(
      async () => await CancelAsync());
  }
  #endregion

  #region SaveAsync Method
  public override async
    Task<EntityLayer.Color?> SaveAsync()
  {
    EntityLayer.Color? ret
      = await base.SaveAsync();

    if (ret != null) {
      await Shell.Current.GoToAsync("..");
    }

    return ret;
  }
  #endregion

  #region CancelAsync Method
  public async Task CancelAsync()
  {
    await Shell.Current.GoToAsync("..");
  }
  #endregion

  #region EditAsync Method
  protected async Task EditAsync(int id)
  {
    await Shell.Current.GoToAsync(
      $"{nameof(Views.ColorDetailView)}?id={id}");
  }
  #endregion
}
```

Create a Color View Model

Now that you have the entity and repository classes for colors created, create a color view model class. Right mouse-click on the **ViewModelClasses** folder in the **AdventureWorks.ViewModelLayer** project and add a new class named **ColorViewModel**. Replace the entire contents of this new file with the code shown in **Listing 16**.

You now need a color view model class to handle commanding in .NET MAUI. Right mouse-click on the **MauiViewModelClasses** folder and add a new class named **ColorViewModel**. Replace the entire contents of this new file with the code shown in **Listing 17**.

Create a Color Detail View Page

After displaying a list of colors, you should also allow the user to add or edit colors by creating a color detail page. Right mouse-click on the Views folder and select **Add > New Item...** and then **.NET MAUI > .NET MAUI ContentPage (XAML)**. Set the Name to **ColorDetailView** and click the Add button. Set the **Title** attribute to **Color Information**. On the **Views\ColorDetailView.xaml** file, add a couple XML namespaces to the partial views' namespace and the view model namespace.

```
xmlns:partial="clr-namespace:
  AdventureWorks.MAUI.ViewsPartial"
xmlns:vm="clr-namespace:
  AdventureWorks.MAUI.MauiViewModelClasses"
```

Listing 18: Create a color detail view to edit colors.

```
<Border Style="{StaticResource Border.Page}">
  <ScrollView>
    <Grid Grid.Row="0"
          Grid.Column="1"
          RowDefinitions="Auto,Auto,Auto,Auto,Auto"
          ColumnDefinitions="Auto,*"
          Style="{StaticResource Grid.Page}">

      <partial:HeaderView Grid.Row="0"
        Grid.ColumnSpan="2"
        ViewTitle="Color Information"
        ViewDescription="Use this screen
          to modify color information." />

      <Label Grid.Row="1"
             Text="Color Name" />
      <Entry Grid.Row="1"
             Grid.Column="1"
             Text="{Binding
               CurrentEntity.ColorName}" />

      <HorizontalStackLayout Grid.Row="2"
                             Grid.Column="1">
        <Button Text="Save"
          ImageSource="save.png"
          ToolTipProperties.Text="Save Data"
          ContentLayout="Left"
          Command="{Binding SaveCommand}" />
        <Button Text="Cancel"
          ImageSource="cancel.png"
          ContentLayout="Left"
          ToolTipProperties.Text="Cancel Changes"
          Command="{Binding CancelCommand}" />
      </HorizontalStackLayout>
    </Grid>
  </ScrollView>
</Border>
```

Add an x:DataType attribute to the page to take advantage of compiled bindings.

```
x:DataType="vm:ColorViewModel"
```

Replace the <HorizontalStackLayout> element with the code shown in **Listing 18**. You don't need to add or edit the **ColorId** property from the Color class, so just add a single Label and an Entry control for the **ColorName** property.

Open the **Views\ColorDetailView.xaml.cs** file and replace the entire contents of this file with the code shown in **Listing 19**. This code should look very familiar by now, as it follows the same design pattern as the one established in the user and product detail pages.

Modify AppShell

The color list page is called from a ShellContent element in the AppShell, so it's already registered in route navigation. However, you need to register the color detail page, as it's not referenced anywhere in XAML. Open the **AppShell.xaml.cs** file and register the route to the ColorDetailView in the constructor.

```
Routing.RegisterRoute(
  nameof(Views.ColorDetailView),
  typeof(Views.ColorDetailView));
```

Modify the Color List Page

Let's fix up the color list page you added before and use the CarouselView control. Open the **Views\ColorListView** file. Add XML namespaces for the commanding view models, entity layer, and partial views, as shown in the following code:

```
xmlns:partial="clr-namespace:
  AdventureWorks.MAUI.ViewsPartial"
xmlns:vm="clr-namespace:
  AdventureWorks.MAUI.MauiViewModelClasses"
xmlns:model="clr-namespace:
  AdventureWorks.EntityLayer;
    assembly=AdventureWorks.EntityLayer"
```

Add an **x:DataType** attribute to the <ContentPage.Resources> element to take advantage of compiled data bindings.

```
x:DataType="vm:ColorViewModel"
```

Add an x:Name attribute so the edit button on the list can reference back to the page to call the EditCommand property in the color view model.

```
x:Name="ColorListPage"
```

When you created the color list page, you added a single <Label> element to display text that described this page. Replace the <Label> element with the following XAML.

Listing 19: Add code to display a color on the detail page.

```
using AdventureWorks.MAUI.MauiViewModelClasses;

namespace AdventureWorks.MAUI.Views;

[QueryProperty(nameof(ColorId), "id")]
public partial class ColorDetailView
  : ContentPage {
  public ColorDetailView(
    ColorViewModel viewModel) {
    InitializeComponent();

    _ViewModel = viewModel;
  }

  private readonly ColorViewModel _ViewModel;
  public int ColorId { get; set; }

  protected async override void OnAppearing() {
    base.OnAppearing();

    // Set the BindingContext to the ViewModel
    BindingContext = _ViewModel;

    // Retrieve a Color
    await _ViewModel.GetAsync(ColorId);
  }
}
```

Listing 20: Use a CarouselView, a Frame, and an IndicatorView for small lists of items.

```
<CarouselView Grid.Row="1"
   IndicatorView="colorIndicators"
   ItemsSource="{Binding Path=Colors}">
  <CarouselView.ItemTemplate>
    <DataTemplate x:DataType="model:Color">
      <StackLayout>
        <Frame HasShadow="True"
          BorderColor="DarkGray"
          CornerRadius="5"
          HorizontalOptions="Center"
          VerticalOptions="CenterAndExpand">
          <Grid Margin="20"
            RowDefinitions="Auto,Auto"
            ColumnDefinitions="Auto,*"
            RowSpacing="20">
            <Rectangle Grid.Row="0"
              Grid.Column="0"
              BackgroundColor="{Binding ColorName}"
              WidthRequest="40"
              HeightRequest="40"
              Margin="10" />
            <Label Grid.Row="0"
              Grid.Column="1"
              FontSize="Large"
              Text="{Binding ColorName}" />
            <HorizontalStackLayout Grid.Row="1"
               Grid.ColumnSpan="2">
              <ImageButton Source="edit.png"
                ToolTipProperties.Text="Edit Color"
                CommandParameter="{Binding ColorId}"
                Command="{Binding
                  Source={x:Reference ColorListPage},
                  Path=BindingContext.EditCommand}" />
              <ImageButton Source="trash.png"
                ToolTipProperties.Text="Delete Color" />
            </HorizontalStackLayout>
          </Grid>
        </Frame>
      </StackLayout>
    </DataTemplate>
  </CarouselView.ItemTemplate>
</CarouselView>

<IndicatorView Grid.Row="2"
               x:Name="colorIndicators"
               Margin="5"
               IndicatorSize="20"
               IndicatorColor="LightGray"
               SelectedIndicatorColor="DarkGray"
               HorizontalOptions="Center" />
```

Listing 21: Load the colors for displaying in the CarouselView.

```
using AdventureWorks.MAUI.MauiViewModelClasses;

namespace AdventureWorks.MAUI.Views;

public partial class ColorListView : ContentPage {
  public ColorListView(ColorViewModel viewModel) {
    InitializeComponent();

    _ViewModel = viewModel;
  }

  private readonly ColorViewModel _ViewModel;

  protected async override void OnAppearing() {
    base.OnAppearing();

    BindingContext = _ViewModel;

    await _ViewModel.GetAsync();
  }
}
```

```
<Border Style="{StaticResource Screen.Border}">
  <Grid Padding="20">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>

    <partial:HeaderView
       ViewTitle="Color List"
       ViewDescription="The list of colors
         in the system." />

  </Grid>
</Border>
```

Just after the <partial:HeaderView...> element, add the XAML shown in **Listing 20**. The CarouselView control has an **IndicatorView** property to which you set the name of the IndicatorView control. As you can see, the x:Name attribute defined on the IndicatorView is set to **colorIndicators**, thus the **IndicatorView** property on the CarouselView control is set to this same name.

As you swipe through the CarouselView, or if you click a dot on the IndicatorView, the same element in the collection is selected, thus updating the other control. In the DataTemplate for the CarouselView a <Rectangle> element is used to display the color name as an actual color on the screen (as you saw in **Figure 7**). This is accomplished by binding the BackgroundColor property to the **ColorName** property on the Color object being displayed.

To load the colors, pass the ColorViewModel class to the color list page. Open the **ColorListView.xaml.cs** file and make it look like the code in **Listing 21**.

Open the **ExtensionClasses\ServiceExtensions.cs** file and in the AddRepositoryClasses() method, add the ColorRepository to the DI container.

```
services.AddScoped<IRepository<EntityLayer.Color>,
   ColorRepository>();
```

In the AddViewModelClasses() method, add the ColorViewModel to the DI container.

```
services.AddScoped<MauiViewModelClasses
  .ColorViewModel>();
```

In the AddViewClasses() method, add the ColorListView and ColorDetailView to the DI container.

```
services.AddScoped<ColorDetailView>();
services.AddScoped<ColorListView>();
```

Try It Out

Within Visual Studio, switch to the **Android Emulator** and run the application. Click on the **Maintenance > Colors** menu. Swipe through the colors to see the IndicatorView control update and view the different colors, as shown in **Figure 7**. Click on the edit button to view the color detail page.

Summary

In this article, you used different list controls available in .NET MAUI to display a collection of data. Use the ObservableCollection class for collections as any changes made to the collection are immediately reflected in any bound list controls. The ListView control has a few different ways to present data, but the most flexible is to use the ViewCell. The CollectionView control is the preferred control to use as it's the most flexible and performant of any of the list controls. If you have a small set of data, you might use the CarouselView combined with the IndicatorView. However, be aware that currently, this control is only optimized to work on the Android and iOS platforms. Coming up in the next article, you'll learn to display information and error messages, display pop-up dialogs, and to validate data in .NET MAUI.

Paul D. Sheriff

SPONSORED SIDEBAR:

Ready to Modernize a Legacy App?

Need advice on migrating yesterday's legacy applications to today's modern platforms?

Take advantage of CODE Consulting's years of experience and contact us today to schedule a FREE consulting call to discuss your options.

No strings.
No commitment.

For more information: www.codemag.com/consulting or email us at info@codemag.com.

Career Development and Staffing reinvented

You think great talent and cool positions only exist in Silicon Valley? Think again!

CODE Staffing is a high-tech, career development, and - for the lack of a better term - "staffing company," that provides a modern workplace with the number one goal being the development of talent and careers. CODE Staffing was started to disrupt the traditional staffing industry. We provide an environment that attracts top talent. We offer unrivaled benefits and a cool workplace culture with unlimited upward career development opportunities.

Many companies think they cannot provide a work environment that rivals the "cool Silicon Valley startups," and many employees think they won't be able to find such a workplace outside "The Valley." We have changed that equation!

Our Mission

CODE Staffing delivers unparalleled contingent technical staffing solutions to our clients. We empower their businesses with the talent and expertise they need to thrive. We focus on long-lasting partnerships based on trust, transparency, collaboration and by providing access to our extensive network of industry connections.

We provide a great environment for our employees so that we minimize churn for our clients. Our rigorous focus on training and regular meetings with clients' executives ensure that we understand their technological road map and guarantees that we provide the best people for their needs.

At CODE Staffing, we're committed to delivering exceptional value to our clients, providing a fulfilling work experience for our employees, and building long-lasting partnerships within our community.

Our Vision

To achieve our mission, we focus on two core groups: our clients and our people.

For Our Clients

Our business philosophy is centered on building strong, long-lasting partnerships with our clients, rather than pursuing a large number of short-term engagements. By fostering close relationships with the organizations we serve, we aim to become a trusted and integral part of their operations, and to support their growth and success over the short and long term. We are committed to working collaboratively and transparently with our clients to enable their teams to deliver continued value to their businesses.

Our clients want to be able to focus on their business, not on managing multiple vendors for staffing. Here are some of the ways we help our clients build better software...

We act as a single vendor that knows our clients' business.

Prioritizing a few key accounts rather than a multitude of small accounts allows us to provide personalized services tailored to each of our clients. Our team works closely with our clients' organizations to gain a thorough understanding of their intricacies and to better grasp their overall goals and objectives.

We act as trusted industry advisors to our clients.

Our executive team at CODE is a group of technologists who have owned long-standing, successful businesses. They have experience growing technology teams to thousands of people and have learned that the only way to succeed is to invest in our own employees. Every key client is assigned a member of our executive team as their contact – with regular quarterly meetings to understand where they are heading – both as a business and with technology. This approach enables us to ensure that our staff receives training in the technologies that our clients are moving towards, ensuring that we are always prepared to meet their needs.

We are committed to training and adopting new technologies in a straightforward manner.

Our dedication to keeping our staff well-trained and informed on the latest technologies ensures they are fully equipped to tackle any project for our clients. Our commitment to training paired with our vast experience in the field gives our team the confidence they need to excel in every client engagement while staying up to date on the latest technology.

We leverage CODE Training, a division of the CODE group, to train not only our employees in upcoming technologies, but we make the training available to our clients' personnel as well. By doing so, we enable their personnel to concentrate on their business operations, while we take charge of training both our own and our clients' employees in technology.

We provide a clear legal differentiation between clients' employees and contractors, offering more flexibility.

The contractors we provide are our employees – with salary and benefits, as well as regular training. CODE takes the heavy lifting out of managing (sometimes hundreds of) contractors, leaving our clients more time to focus on their projects and growing their business.

We create a system to keep contractors with our clients for the long term.

By employing great developers and providing ownership via profit sharing, CODE Staffing employees are incentivized to stay with us and our clients for the long run. Focusing on training and the long-term investment in top talent not only helps the organizations we work with achieve sustained success but ensures that institutional knowledge doesn't regularly disappear.

For Our People

CODE Staffing is centered around creating an exceptional workplace environment that attracts and retains top talent, enabling us to provide our clients with a stable and reliable team. We prioritize fulfilling our employees' essential needs by providing engaging work, competitive compensation, a sense of ownership and pride, and continuous opportunities for growth in both their technical and interpersonal skills.

Similar to our business philosophy, we prioritize long-term investment in our staff over filling short-term positions. We are committed to developing the full potential of our employees and supporting their career growth over time. To this end, we provide career mentorship opportunities, generous salaries and benefits, a profit-sharing plan that incentivizes everyone to work toward a common goal, and continuous comprehensive training in the latest technologies as well as in the industries our clients are focused on.

At CODE Staffing, we believe that happy employees lead to happy clients. By prioritizing employee satisfaction and delivering outstanding results for our clients, we can build strong, lasting partnerships based on trust, transparency, and exceptional service.

CODE

Content paid for by codestaffing.com

ONLINE QUICK ID 2503031

Offline AI Image Generation

On January 5, 2021, OpenAI revealed DALL-E. Frankly, it blew everyone's minds. DALL-E was a modified version of GPT-3. GPT, as you might know, is a large language model (LLM), and it generates text. But DALL-E took an input prompt and generated an image out of it. No, not searched for images across the internet, but generated a brand-new image on the fly. The demand was great—everyone had to try it. There was a wait list. It took a while to generate an image and the images weren't that great, to be honest. Fast-forward barely three years later, and you have many such text-to-image models. DALL-E has undergone a few versions already. And now there's Midjourney, Stable Diffusion, Artbreeder, Deep Dream Generator, Prisma, Craiyon, starryAI, and many more. The images have gotten so good that they're frequently indistinguishable from real images. This has created some strange problems; the phrase "seeing is believing" is no longer true.

Sahil Malik
www.winsmarts.com
@sahilmalik

Sahil Malik is a Microsoft MVP, INETA speaker, a .NET author, consultant, and trainer.

Sahil loves interacting with fellow geeks in real time. His talks and trainings are full of humor and practical nuggets.

His areas of expertise are cross-platform Mobile app development, Microsoft anything, and security and identity.

In fact, these models have gotten so good and so accessible that you can run them offline, on a commercial off-the-shelf computer. I'm writing this article on a shiny new M4 Max, and I've tested the code on an M1 Max. It's quite realistic to generate images using text prompts with commonly available models on your laptop. By the end of this article, you'll be able to do so on your machine as well.

Pardon me, but I can barely contain my excitement. Can you imagine what this means? My thoughts turn to reality on-the-fly and I'll be the best meme maker in town. All right, more than just that. When building presentations, I could just fire up my local AI model, type in a text prompt, and have it generate a nice image for me. Why doesn't PowerPoint have such a feature? Maybe it does—if not, I'm sure that, at some point, it will.

Of course, this raises eyebrows and numerous ethical concerns. How do the artists whose images were used to create these models get paid? What implication does this have on what is trustable and what is not? Can a quick tweet on an AI-generated image create a news headline and start a war? Can images that look absolutely real be used to change the course of wars? Can you turn stills into a video, appear in a fake webcam stream, and appear as someone you're not? What does this mean for social engineering? What does this mean for human relationships? Will we all have AI generated boyfriend/girlfriends or just friends backed by an LLM, glued to our VR headsets?

Sadly, all those are absolutely real concerns, concerns that have already manifested themselves in many ways. And, as local hardware has gotten so powerful, all this is well within the reach of anyone with a credit card. Ouch!

I feel the best solution here would be a blockchain plus certificate-based distributed image verification solution, so at least we'd know if an image is trustable. Trust me, looking at an image and trusting that it's real because it's a photo is no longer enough. Don't believe me? Visit ThisPersonDoesNotExist.com and let me know what you think.

If I had enough video or picture data of you, could I generate a model of you, and basically generate any image of you, pretty much indistinguishable from reality? Sadly yes!

All right, it's best if I focus on the tech aspects of this, so back to that. In this article, I'm going to show you a few examples of applications that take in a text prompt and generate an image for you. Although I'll depend on open source or openly available models, the actual execution will be local, on my Mac. The code should be able to run with no internet connection, entirely on your laptop, as long as you have some beefy hardware. Either a higher-end MacBook pro, or a Windows or Linux machine with a high-end Nvidia card will do.

Why Offline?

You might be wondering: Why bother doing this offline? There are models, such as Midjourney, that are available as a service for as low as $10 a month. Why not just use them?

There are some very good reasons to do this offline. First, I'm cheap. I don't want to pay $10 a month. If you do this seriously at a commercial scale, it'll cost you more than that. But either way, being cheap is a pretty good reason.

Second, security and privacy are important. Your data stays local, so you don't have to upload any sensitive information anywhere. What you generate is your business. An offshoot of text-to-image generation is image-to-image generation, where you can upload an image that you have and use that as an inspiration to generate more images. You wouldn't want to upload your images to some random service whose data residency requirements don't agree with your terms, right? You do read those terms, right?

Third, the ability to run this offline means you can run this anywhere. Afterall, the ping times between Earth and Mars are anywhere between eight and 16 minutes. I wouldn't want to wait eight minutes for an image to generate only to realize that I want to tweak it and generate another image and wait another eight minutes. Jokes aside, image generation is still compute-intensive, given today's hardware. Typically, when we're trying to get the perfect image, we input and experiment with many parameters, and prompts, and refiners, to get the image just right. It takes iterations, and we may want to move fast with a lower resolution picture and then increase the resolution as we get closer to the result we want. Any serious image generation shop will need something like this.

The code I'll show you runs on my local M4 Max, and each image is generated within seconds. You could easily chain up a bunch of M4 Pros, for example, or similar Nvidia-based hardware in a cheap server farm, and accelerate this process for real-world applications. Imagine if you had a creative team on staff. They could issue prompts and generate images on the fly, supporting a team of five to ten people, on a server farm costing maybe USD 10K, and very low power consumption. This is the realistic value you can unlock with today's hardware and I have no doubts it'll only get better.

The fourth reason is creative freedom. Many online image-generation AI models have been accused of bias. I won't point out specific reasons, but let's just agree that bias exists. So does censorship. What if I want to generate images that suit my needs and my taste without some big company (or even government) telling me what I can or cannot do. All right, please don't get me in trouble on this, or get yourself in trouble on this, but if you generate problematic or misleading content, it still wouldn't be cool, and you will probably face the consequences for it. But still, let's say you were directing a film based on World War II, and you needed to generate an image of Germany during 1942. Obviously, such an image would have questionable artifacts that perhaps many online solutions will block for good reason. But for your use case, you should be able to generate something like this offline, right? As long as it's put to a moral use. Unfortunately, there are plenty of NSFW models available easily as well, but we'll keep it work friendly, all right?

The fifth reason is commercial applicability. Many online-generated images are not quite what you're looking for. Either their image styles are too generic, or they have their own personality and therefore aren't unique enough for your brand, or maybe they're watermarked, or come with restrictions, or they're not sufficient resolution for the specific task you're attempting. What if you wanted to print a poster, and you needed a 6000px by 6000px AI-generated image?

Okay, let's think something crazy. What if you were directing a film based in the year 48288 AD, and you wanted to generate a futuristic landscape projected on curved super high-resolution screens. Imagine the cost savings of being able to generate an AI image and project it on a screen that surrounds the actors. The actors wouldn't need to even pretend or act. And it'll save so much editing time. No more green screens, no more expensive sets, but you will need super high-resolution images.

Finally, you may want to fine-tune an image generation model with your own images. Imagine you're a Hollywood producer, and for some exceptionally risky scenes, you don't want to risk even a stunt man. Could you create an AI version of an extremely overpaid Hollywood actor, and have the AI version do all the dangerous stuff? This requires you to create an AI model specific to that actor. What would that mean for news readers? See this video https://x.com/andrewrsorkin/status/1856849559756181904. Let me know what you think.

Frankly, why do we even need actors at that point?

There are many other such reasons. But let's just say that enough reasons exist to try this. So let's try it.

Text-to-Image Models

The specific kinds of models you'll use for this article are text-to-image models. As the name suggests, you give it an input text prompt and it'll generate an image for you. In contrast, there are image-to-image models as well, where you can start by using an existing image and making tweaks to it. For instance, you could generate a cartoon version of my mugshot.

Let's understand a few basic concepts. First, what is a diffusion model? A diffusion model is a type of generative model that learns to represent data as a Markov chain that progressively adds noise to the input data until it becomes a random sample from a known distribution (e.g., Gaussian noise). The model then learns to reverse this diffusion process to generate new samples that are similar to the original data. So, between forward diffusion, denoising, and reverse diffusion, it's able to help you do things like image generation, image-to-image translation, or data manipulation, like adding missing details into a picture. Text-to-image models, like latent diffusion models, combine an LLM that transforms an input text into a latent representation and a generative image model, which is then able to generate an image.

As it turns out, there are many AI models that can run offline, such as Stable Diffusion, DALL-E Mini (Craiyon), a limited version of Midjourney, etc. Some of these models are available as applications, and sometimes those apps work only on mobile devices. I wanted to pick a model that gave me the most freedom. My requirements were that I wanted the most flexibility in picking the image generation model. Maybe even extend it with more models. I wanted the ability to fine-tune the model with my images. I wanted to use a model that could generate incredibly high-resolution images, and I wanted to do this and much more through custom code.

What I really wanted was my own web application where I can input a text prompt and generate an image within my corporate network (or, for now, my home network). Let's see how far we can take this.

After some homework, I decided to go with Stable Diffusion for this article. I'm not saying this is the best possible model—there were so many other choices at https://huggingface.co/models?pipeline_tag=text-to-image&sort=trending. But after some high-level research, it seems that Stable Diffusion comes in various sizes so I could size it to my hardware. I don't have a server farm, just one laptop, so I need to be able to run this low for dev purposes and scale it high for production. I wanted a model that has decent support and community interest, that performs reasonably fast on reasonable hardware, and that allows me to tweak input enough to generate exactly what I wish. Most of all, I wanted it to be free, which Stable Diffusion seems to be.

Stable Diffusion

Stable Diffusion is a deep learning text-to-image model. It's built by a company called Stability AI. It can be used for text-to-image applications, but also for other scenarios, such as image-to-image, inpainting, outpainting, etc. Stable Diffusion was trained on an open-source dataset of images at www.laion.ai. Given that the input dataset had relatively lower resolution images of 512x512, Stable Diffusion excels at generating images around that resolution. In real-world applications, you can use Stable Diffusion to generate such images, and once you're happy with the image, you can use one of many upscaling models to get a higher-resolution image if you wish. Although I won't cover upscaling in this article, you're welcome to check out https://openmodeldb.info/models/4x-Nomos8kDAT as one of my favorite upscaling models. As subsequent versions of Stable Diffusion have rolled out, Stable Diffusion can generate good images at 1024x1024.

Stable Diffusion may also suffer from not being able to generate certain details accurately. Notably, it may inaccurately generate limbs. You can avoid this, to a great extent, by using the right prompts. But it's also not uncommon to use refiner models to get further details on certain aspects of the image. For instance, you can use dedicated models to refine skin to make it look more realistic, or face features, or limbs, or really anything.

A Simple Image Generation Application

Okay, enough talk. Let's write some code. The model I decided to go with for my local application can be found at https://huggingface.co/stabilityai/stable-diffusion-3-medium. To use this model, you'll have to create an account at Hugging Face, agree to the terms of usage, and share your contact information. I'm no lawyer, but it seems like this model is free for research, non-commercial, or even commercial usage, as long as your revenue is less than $1M per year. I think I'm pretty safe there. You can read their license here https://stability.ai/license. If your revenue is > $1M, well, congratulations. You can reach out to Stability AI for a commercial enterprise license.

Listing 1: The simple image generation code

```
import torch
from diffusers import StableDiffusion3Pipeline

pipe = StableDiffusion3Pipeline.from_pretrained(
    "stabilityai/stable-diffusion-3-medium-diffusers",
    torch_dtype=torch.float16)
pipe.to("mps")

image = pipe(
    prompt="A cartoon image of yellow dog
    sitting on a chair, drinking coffee
    with fire all around him,
    saying this is fine",
    negative_prompt="",
    num_inference_steps=28,
    height=512,
    width=512,
    guidance_scale=7.0,
).images[0]

image.save("output.png")
```

Listing 2: The launch.json for debugging support

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python Debugger: Current File",
            "type": "debugpy",
            "request": "launch",
            "program": "${file}",
            "console": "integratedTerminal"
        }
    ]
}
```

I intend to use Stable Diffusion with diffusers. Diffusers are a type of neural network component used in deep learning models, particularly in the field of computer vision and image processing. They're a key component of diffusion models, which I mentioned earlier. Diffusers are used in image generation models, such as generative adversarial networks (GANs) and variational autoencoders (VAEs), to generate new images.

To get started, set up a Python project, with a virtual environment targeting Python 3x. Go ahead and create a requirements.txt with the following code in it.

```
torch
diffusers
transformers
protobuf
sentencepiece
accelerate
```

Then, in your .venv terminal, run the following command to install the necessary packages.

```
pip install -r requirements.txt
```

Next, create a file called **index.py** with the code shown in **Listing 1**. Let's understand this code a bit better.

The first thing you do is create a diffusion pipeline using StableDiffusion3Pipeline.from_pretrained. A diffusion pipeline is a series of processing steps used in diffusion models to progressively refine and generate data, such as images. You're creating one from a pre-trained model here. Could you use other pre-trained models? More on that later.

> A diffusion pipeline is a series of processing steps used in diffusion models to progressively refine and generate data, such as images.

Next, use the pipeline to generate an image. When you generate an image, you can pass in several parameters, many of them optional. Here's where you have to play around with the right prompts and parameters to get the right result. At the very least, you need to give it a prompt, i.e., what you're asking it to generate an image of. You can also pass in negative prompts, which are instructions to the model to avoid generating certain kinds of images. For example, a negative prompt could be "No gore," to avoid gory images. Then you have num_inference_steps, which is the number of denoising steps. More denoising steps usually mean a better-quality image at the expense of slower inference. Height and width are pretty obvious parameters. And finally, there's guidance_scale. Higher guidance scale encourages it to generate images that are closely linked to the text prompt, usually at the expense of lower image quality.

Finally, to get debugging to work in VSCode, create a .vscode folder and create a launch.json file with the code shown in **Listing 2**.

That's basically it. Now hit F5, and, in almost no time, you should see an image generated in the output.png file.

The prompt I used was:

> A cartoon image of yellow dog sitting on a chair, drinking coffee with fire all around him, saying "This is fine."

The image I was able to generate can be seen in **Figure 1**.

I LOL'ed big time when I saw that picture. I mean, look at how the dog is holding the cup. Okay, AI did its best. Seriously, how else would a dog hold a cup? That's a bit like how a dog would wear pants right? All four legs or only rear paws? Also, dogs don't drink coffee—coffee is poisonous for dogs.

Let's just agree that this code is impressive, but also, that this is not so fine and it needs improvement. Can our very simple model generate somewhat realistic pictures?

Change the prompt to:

> Smiling woman in turquoise silk curtains. Fall sunlight. Professional photography.

I got the image you can see in **Figure 2**.

My first reaction: ehh! Too bad it's just AI!

Yeah, the picture looks a bit more photo-realistic, and I can sort of tell it is a person. But what's up with her left arm? The face looks a bit deformed. The skin looks artificially smooth and those curtains look haphazardly arranged. It's impressive for a computer program to generate this, but I'm not fooling anyone with this. It's generated by AI. It almost looks like a poor photoshop. You can keep playing with the prompts, inference steps, guidance, etc., but I wasn't able to produce an image that I could show someone without them being able to tell whether this is AI or not. In fact, try generating pictures of celebrities, like Satya Nadella or Taylor Swift. It generates a picture that kind of looks similar to them, but it's clearly not a picture of the person in question. Maybe that's by design. After all, we just went through an election in the U.S., and there were plenty of AI-generated pictures floating around. Thankfully, for most of them, you could tell they are just AI.

I want to explore this a bit further, so I headed over to https://civitai.com.

Civitai is a website where users can share models, images, videos, articles, and all-around image and video generation. There are many models to pick from and many of them are uncensored. Contributors share their work and frequently explain how they arrived at a certain result.

I could take an existing model and fine tune it, but a lot of that work has already been done for me in the various models available in Civitai. As you explore Civitai, you'll

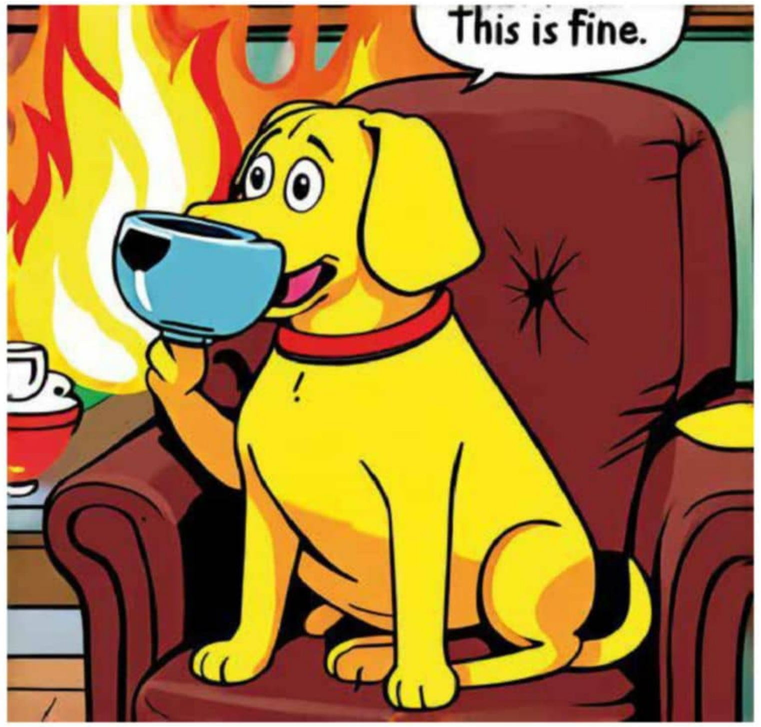

This is fine.

Figure 1: This is fine.

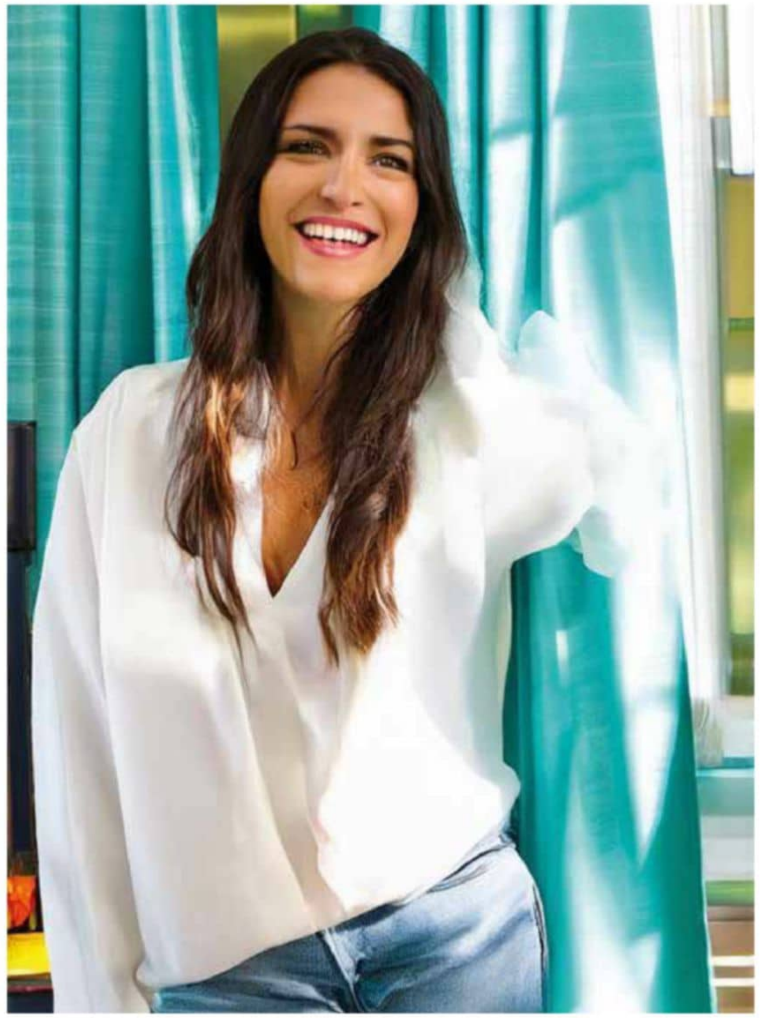
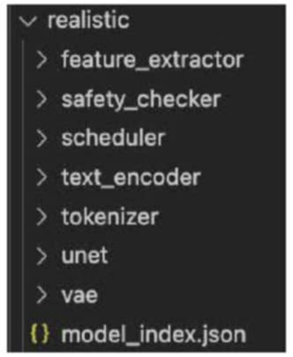


Figure 2: Trying to produce a more realistic image



```
realistic
  feature_extractor
  safety_checker
  scheduler
  text_encoder
  tokenizer
  unet
  vae
  model_index.json
```

Figure 3: Safetensors converted to diffusers

find models that are appropriate for various purposes. Some are trained on a particular celebrity. Some are great for generating scenery. My goal was to generate realistic images, so this AI model (of two people) caught my eye: https://civitai.com/models/4201/realistic-vision-v60-b1. The same models are also available on Hugging Face if you prefer.

Although you can download the model directly from Civitai, in order to use it with a diffusion pipeline, it's going to need some prep work.

First go ahead and download the model of your choice. In my case, I downloaded realistic-vision v5.1 Hyper (VAE). This downloaded model is a safetensors file. You need to convert it to a format that can be used with diffusers. Thankfully, Hugging Face provides a script to do that at https://raw.githubusercontent.com/huggingface/diffusers/v0.20.0/scripts/convert_original_stable_diffusion_to_diffusers.py. Download this file and place it in a folder. This file depends on the same Python packages that your project currently does, so I suggest that you create a folder called **Models** in your same project and place this file there.

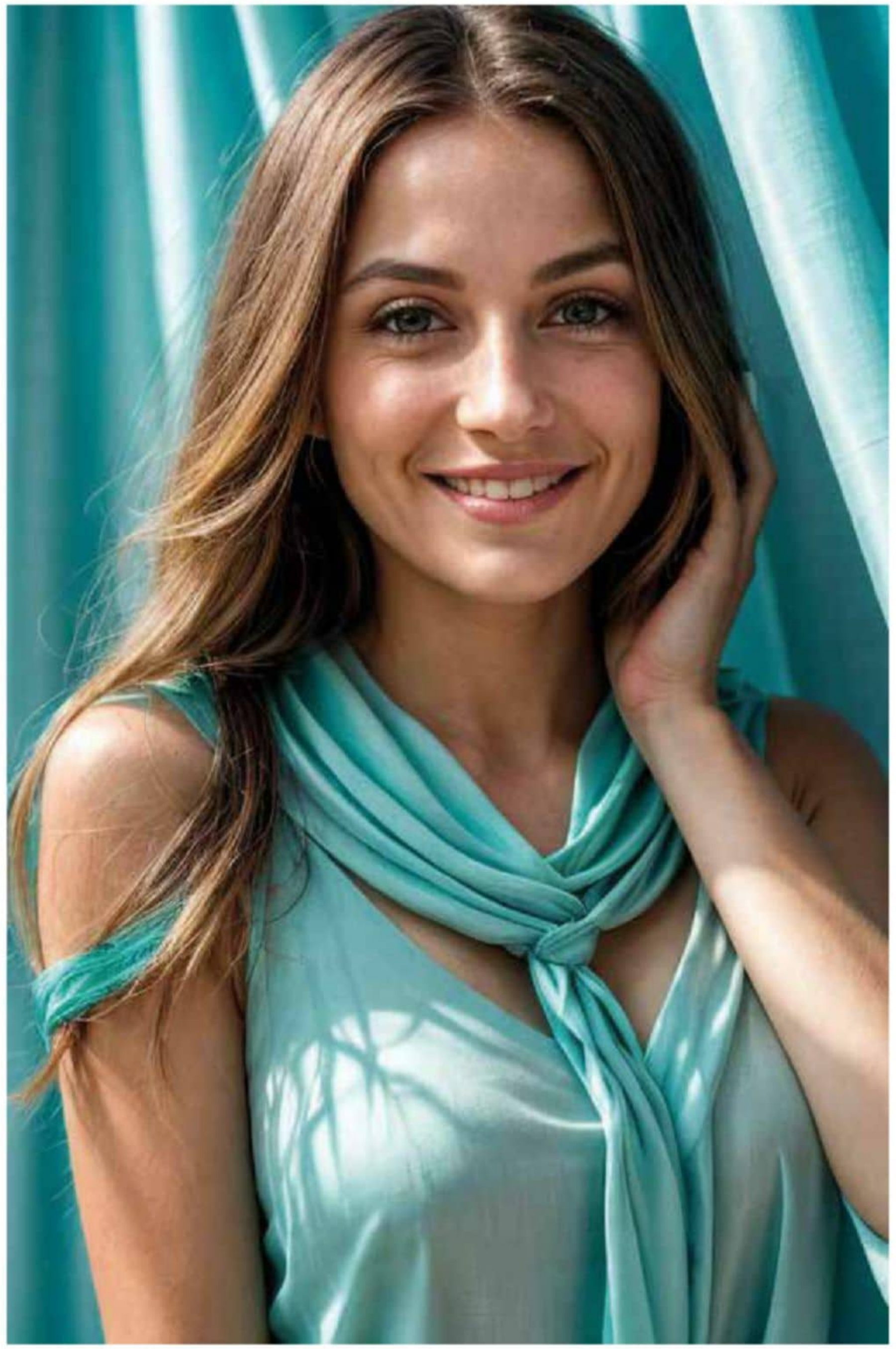


Figure 4: Image generated with the same prompt using realistic model

To convert the safetensors file to diffusers, run the following command:

```
python
convert_original_stable_diffusion_to_diffusers.py
--checkpoint_path
    realisticVisionV60B1_v51HyperVAE.safetensors
--dump_path realistic/
--from_safetensors
```

This will take only a few moments to process. Verify that a folder, as shown in **Figure 3**, now appears in your project under the Models folder.

Great. Now you're ready to consume this model using a diffusion pipeline.

Modify the code as shown in **Listing 3**. Go ahead and run the code. The prompt is same as before:

> Smiling woman in turquoise silk curtains. Fall sunlight. Professional photography.

Now hold your breath. Your image, although similar, may not be identical. The image I was able to generate can be seen in **Figure 4.**

My first reaction: WOW! Too bad it's just AI!

I mean seriously. Does anyone have her number? She doesn't have a ring either. I am frankly blown away at the quality of this image. But even this can get better!

You can choose to upscale this model to a higher resolution. This article is getting a bit long, but you can use an upscaler like https://github.com/zhengchen1999/DAT to upscale this generated image. I went ahead and upscaled this model to 8x, and although the image was still pretty good, I could tell that there was some artifacting going on. **Figure 5** shows it upscaled to 8x and a crop of just her lips.

As you can see in **Figure 5**, the image is still extremely good, but let's be honest, nobody has lips that smooth, especially not in winter. Here you can use extensions for Stable Diffusion, like ADetailer (https://github.com/Bing-su/adetailer), to further work on features such as hands and eyes and so on. With that, you can take a generated image, detect features, and enhance portions of it to be way more realistic.

Remember that earlier I said I was unable to generate photorealistic pictures of celebrities? Maybe that's by design. After all, I don't want to get sued for likeness reasons. But if you're following my article along and have the code working, go ahead and try generating a picture of Taylor Swift. How about Donald Trump? How about Joe Biden? How about Einstein? Now do you believe me? Those pictures look completely indistinguishable from reality. Actually, let me generate a picture of someone who (hopefully) won't mind.

Listing 3: Use the realistic model in a diffusion pipeline

```
import torch
import safetensors
import transformers
import diffusers

model_path = "models/realistic"
pipe = diffusers.DiffusionPipeline.from_pretrained(
  model_path, torch_dtype=torch.float16,
  safety_checker=None, use_safetensors=False)
pipe.to("mps")

image = pipe(
    prompt="
  Smiling woman in turquose silk curtains.
  fall sunlight. professional photography",
    num_inference_steps=30,
    height=768,
    width=512,
    guidance_scale=1.5,
    seed=1876016,
).images[0]

image.save("output1.png")
```

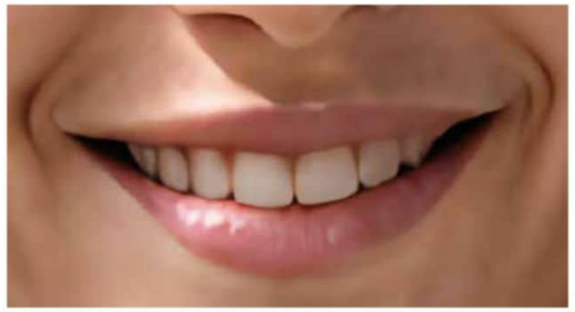


Figure 5: Upscaled to 8x and cropped part of the image.

The prompt I used was:

> Abraham Lincoln riding a motorbike.

The image I was able to generate can be seen in **Figure 6**.

Jaw drop! I had no idea our ex-presidents were such daredevils. But even that image looks like a painting, doesn't it? The issue is that the input data for all pictures of Abraham Lincoln were similar in details. You could fine tune the model and get better results, of course.

Now, I'm not going to push my luck by generating a picture of a modern celebrity who has plenty of pictures available, someone like say Taylor Swift, Donald Trump, or Kamala Harris. But believe me, the generated pictures are frequently indistinguishable from real. And for those who don't have plenty of pictures, you can fine tune with an input set of images.

Putting It All Together

Now that you have the basic code working, can you put this together into a working website where a user can give a prompt and other input parameters, and the server crunches it up to generate a model and show it to the user?

Let's do it!

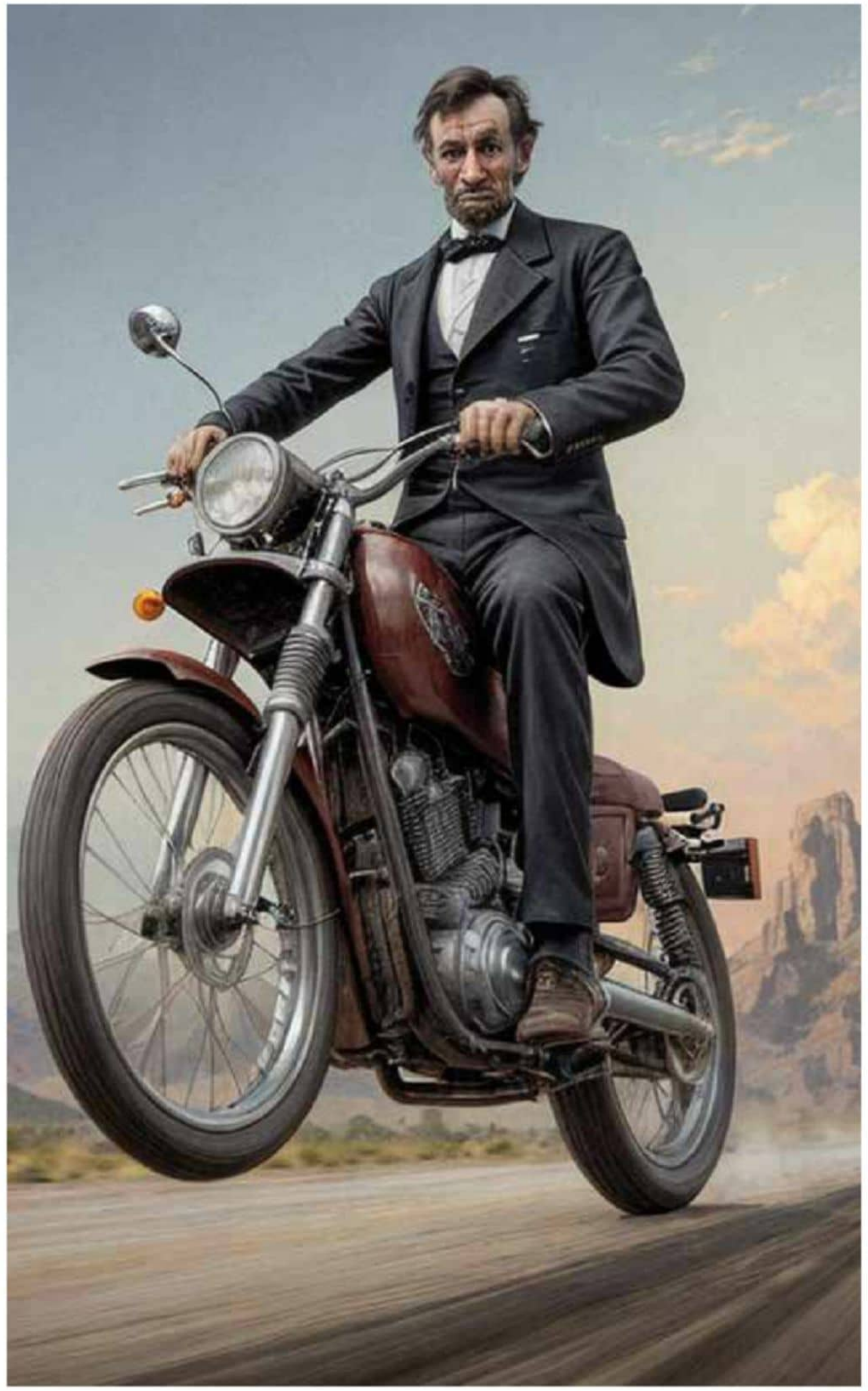


Figure 6: Honest Abe was into motorbikes—maybe!

Listing 4: The index.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Offline AI Image generation</title>
    <script
      src="https://code.jquery.com/jquery-3.6.0.min.js">
     </script>
</head>

<body>
    <h1>Offline AI Image generation</h1>
    Enter prompt:
      <input text="Natural scenery" id="generation-prompt"/>
    <button id="start-generation">Generate Image</button>
    <div id="generation-status"></div>
    <img id="generation-image" src="" />

    <script>
        $(document).ready(function () {
            $('#start-generation').click(function () {
                // Start the generation
                $('#generation-status').text(
                  "generating image");
                $.ajax({
                    type: 'POST',
                    url: '/start_generation',
                    contentType: 'application/json',
                    data: JSON.stringify(
                        {'generation-prompt':
                           $('#generation-prompt').val()}),
                    success: function (data) {
                        $('#generation-status').text(
                            data.message);
                        // Check every 5 seconds
                        let intervalId = setInterval(
                          function () {
                            $.ajax({
                                type: 'GET',
                                url: '/check_generation',
                                success: function (data) {
                                    if (data.message ===
                                         'generation complete') {
                                        // Display the image
                                        $('#generation-image')
                                          .attr('src', data.image);
                                        clearInterval(intervalId);
                                    } else {
                                   $('#generation-status').text(
                                         data.message);
                                    };
                                }
                            });
                        }, 5000);
                    }
                });
            });
        });
    </script>
</body>

</html>
```

Listing 5: Image generation code

```
def generate_image(generation_prompt):
    model_path = "models/realistic"
    pipe = diffusers.DiffusionPipeline.from_pretrained(
       model_path, torch_dtype=torch.float16,
       safety_checker=None, use_safetensors=False)
    pipe.to("mps")

    image = pipe(
        prompt=generation_prompt,
        negative_prompt="",
        num_inference_steps=30,
        height=768,
        width=512,
        guidance_scale=1.5,
        seed=1876016,
    ).images[0]

    image.save("static/image.png")

    # Return the filename
    return 'static/image.png'
```

Listing 6: The start generation route

```
@app.route('/start_generation', methods=['POST'])
def start_generation():
    if os.path.exists('static/image.png'):
        os.remove('static/image.png')
    generation_prompt = request.json['generation-prompt']
    executor.submit(generate_image(generation_prompt))
    return jsonify({'message': 'generating image'})
```

Listing 7: The check generation route

```
@app.route('/check_generation')
def check_generation():
    # Check if the image file exists
    if os.path.exists('static/image.png'):
        return jsonify({'message': 'generation complete',
                        'image': 'static/image.png'})
    else:
        return jsonify({'message': 'generation running'})
```

In the project you were working on, edit the requirements.txt and add the following packages:

```
flask
flask_executor
```

In this same project, create a folder called **Templates** and drop an index.html file in there. Also create a folder called **Static** where generated images will go.

The intention is to allow the user to enter an input prompt and click Generate Image, the server does its AI magic, and when an image is ready, it's shown to the user. Of course, you can make this as compelling as you wish, but to keep things to the point, I won't focus on things like a beautiful user interface, etc.

Let's get started with the index.html first. The index.html is going to leverage jQuery, and it will have a very simple user interface: a text box to receive the user's prompt and a button to start the process of image generation on the server. Then, the code will poll the server every five seconds to see if the image is ready or not. As soon as the image is ready, it will be shown to the user. The full code can be seen in **Listing 4**. Note that I've taken some shortcuts besides the ugly UX. I'm not validating inputs, I'm not allowing the user to specify height/width, etc. But this isn't production code; I'm trying to stay to the point and address things pertinent to the task.

Now let's focus on the server-side code. On the server-side code, you need to turn the console application into a flask application. The idea is that it will serve index.html. It will serve an image.png from the static folder when an image is ready. And it will expose three APIs/routes.

The first route will be **start_generation**, which is callable over POST, as you can see in **Listing 4**. This accepts an input parameter that is the prompt.

The second route will be **check_generation**, which the client will call every five seconds to see if the image is ready.

The third route will be **/image,** where the generated image will be served.

Let's get started.

First, I separated out my image-generation code and parameterized it, as can be seen in **Listing 5**. This is almost exactly the same code as you've already seen in **Listing 3**. The only difference is that the prompt is parameterized. You can make this more flexible by allowing more input parameters, and perhaps even letting the user pick more than one model.

Next is the code to start image generation, as can be seen in **Listing 6**. This code deletes a previously generated file, if it exists, and kick starts the image generation process by calling the **generate_image** function with the user-supplied input prompt. When the image is generated, save it as **static\image.png**.

In **Listing 7**, you can see the **check_generation** route. Here, you simply check to see if the file is present, which means that generation is complete. If the file isn't present, it returns **generation running:** otherwise it returns **generation complete** with the name of the generated image. This code is intentionally simple. In the real world, you'll have an authenticated application, tie this to a user's session, and generate unique filenames for every execution. You might even have a clean-up process to remove images older than a certain date. Those are typical in any production application.

Finally, you have the image route, where you serve the image from the generated file in the static folder. This can be seen in **Listing 8**.

Of course, you'll need a route to serve index.html from the templates folder, which can be seen in **Listing 9**.

The full server-side code put together can be seen in **Listing 10**.

Your project is already setup for debugging, so hit F5 to start debugging. As soon as you hit F5, you should see a message, like below, in VSCode's terminal:

```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 327-338-769
```

Now open the browser and visit the http://127.0.01:5000 URL and you should see a web page, as shown in **Figure 7**.

Ooh, so exciting! Let's enter a prompt. I used the following prompt:

> A cat wearing a Batman suit at sunset, fighting crime in New York, hyper-realistic.

Listing 8: The serving image route

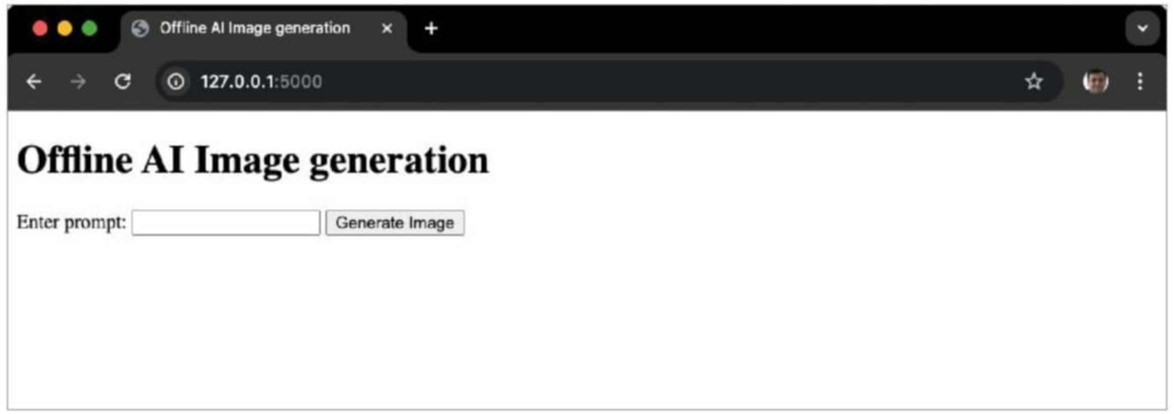
```
@app.route('/image')
def serve_image():
    return url_for('static', filename='image.png')
```

Listing 9: Default route to serve index.html

```
@app.route('/')
def index():
    return render_template('index.html')
```

Listing 10: The full index.py for server side code

```
from flask import Flask,
  render_template, jsonify, url_for, request
from flask_executor import Executor
from PIL import Image, ImageDraw
import os
import torch
import safetensors
import transformers
import diffusers

app = Flask(__name__)
executor = Executor(app)

# Generate the image
def generate_image(generation_prompt):
    model_path = "models/realistic"
    pipe = diffusers.DiffusionPipeline.from_pretrained(
         model_path, torch_dtype=torch.float16,
         safety_checker=None, use_safetensors=False)
    pipe.to("mps")

    image = pipe(
        prompt=generation_prompt,
        negative_prompt="",
        num_inference_steps=30,
        height=768,
        width=512,
        guidance_scale=1.5,
        seed=1876016,
    ).images[0]

    image.save("static/image.png")

    # Return the filename
    return 'static/image.png'

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/start_generation', methods=['POST'])
def start_generation():
    if os.path.exists('static/image.png'):
        os.remove('static/image.png')
    generation_prompt = request.json['generation-prompt']
    executor.submit(generate_image(generation_prompt))
    return jsonify({'message': 'generating image'})

@app.route('/check_generation')
def check_generation():
    # Check if the image file exists
    if os.path.exists('static/image.png'):
        return jsonify({'message': 'generation complete',
                        'image': 'static/image.png'})
    else:
        return jsonify({'message': 'generation running'})

@app.route('/image')
def serve_image():
    return url_for('static', filename='image.png')

if __name__ == '__main__':
    app.run(debug=True)
```



Offline AI Image generation

Enter prompt:

Figure 7: The website is ready.

Creative enough? You feel free to try whatever you wish! My crime fighting kitty can be seen in **Figure 8**.

Haha! This is fun. Let's try another prompt. This time, let's not go realistic. Let's go artistic.

> Watermelon has arms and legs and is running in a cartoon.

The results can be seen in **Figure 9**.

ADVERTISERS INDEX

Blazor, Python, iOS, ChatGPT, Stable Diffusion
CODE
Building Applications with Blazor and ASP.NET

Advertising Sales:
Tammy Ferguson
832-717-4445 ext 26
tammy@codemag.com

Advertisers Index

Advertiser	Website	Page
CODE Consulting--AI Services	www.codemag.com/ai-services	2
CODE Consulting--Executive Briefing	www.codemag.com/executivebriefing	7
CODE Consulting--Copilot	www.codemag.com/copilot	5
CODE Consulting	www.codemag.com/Code	75
CODE Staffing	www.codemag.com/staffing	76
dtSearch	www.dtSearch.com	11
Microsoft Fabric Community Conference	www.fabricConf.com	46

This listing is provided as a courtesy to our readers and advertisers. The publisher assumes no responsibility for errors or omissions.

Honestly, I could play with this all day, and there's so much more you can add here, things I didn't even get a chance to touch upon. For instance, what if you could show the user not one but four generated images from the input prompt? Yep, that's easy: Notice that the generated image is an array. There are four images there, so try it!

Then you can take an image as an inspiration and ask the user to tweak it. For instance, that watermelon looks cool, but turn it into a Picasso painting, and that would be image-to-image generation.

What about chaining multiple models together to perform automated tasks, like enhance certain details, upscale, and well, what about video?

Can I take an input image, and say, turn it into a short video, and give an inspiration to the input? Yeah, all that is possible!

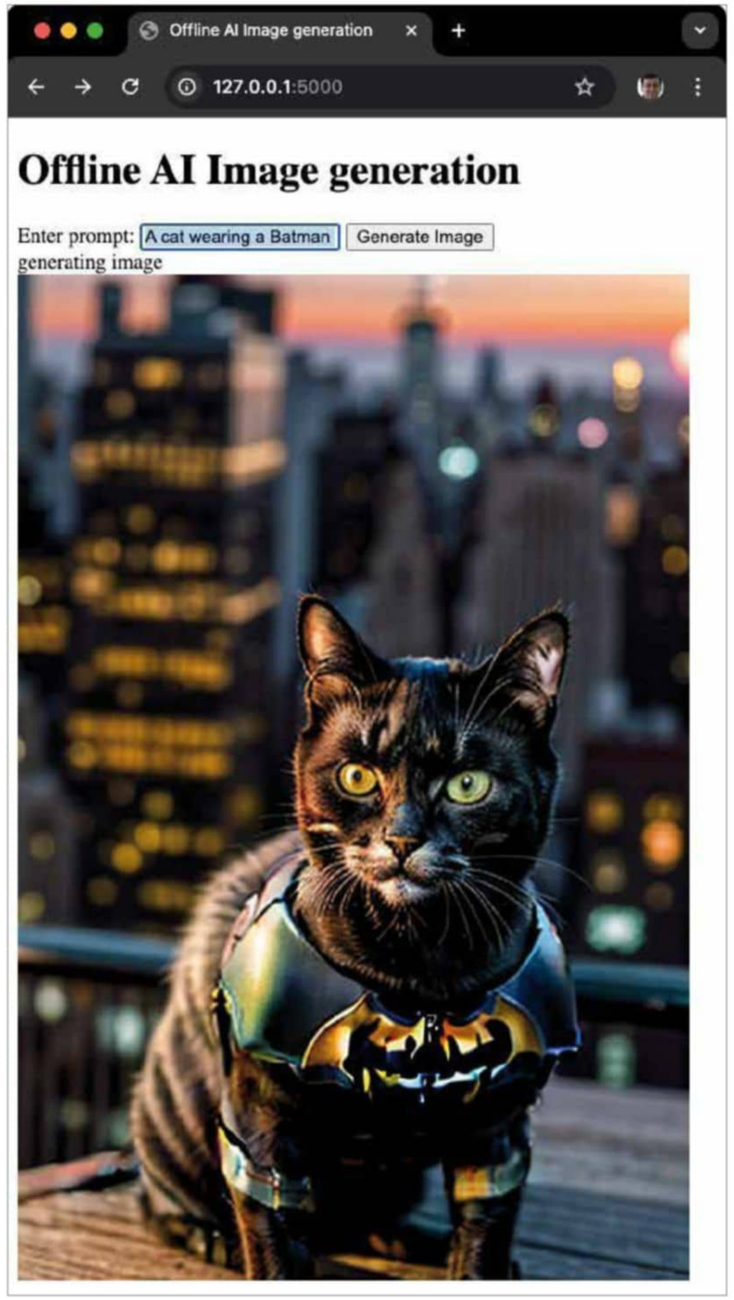
Summary

Here's an interesting fact for you. The most powerful supercomputer in the world is called El Capitan. It cost $600 million to build it, and it needs 28,000 tons of liquid to cool it. It needs 27 megawatts to run it, and I'm sure it has a building and a dedicated staff to baby sit it. In terms of Tera/Exa flops, can you believe that this incredible supercomputer that costs millions and needs the power of a metropolis to run, is only 320 times more powerful than an off the shelf M4 max, which runs on 25 watts of power, exactly the kind I used to write this article?

I could, theoretically speaking, build a server farm of 320 macs for the cost of around 1.6 million USD, and create an AI farm as powerful as the most powerful supercomputer ever built.

Now I know it isn't as simple as that. There's the issue of writing specialized software. Are we entering an era where AI can write this software for me? Believe me, as a race, we are entering an era that is going to blow our minds around what is possible in the hands of mere mortal developers.

I'm serious. Get on the AI bandwagon. Not because you want to be an AI engineer, but because you want to remain relevant. The superpowers you discover will make you 1000x more productive. I'll never bother to learn regex now or internalize complex Git commands. LLMs figure this out for me.

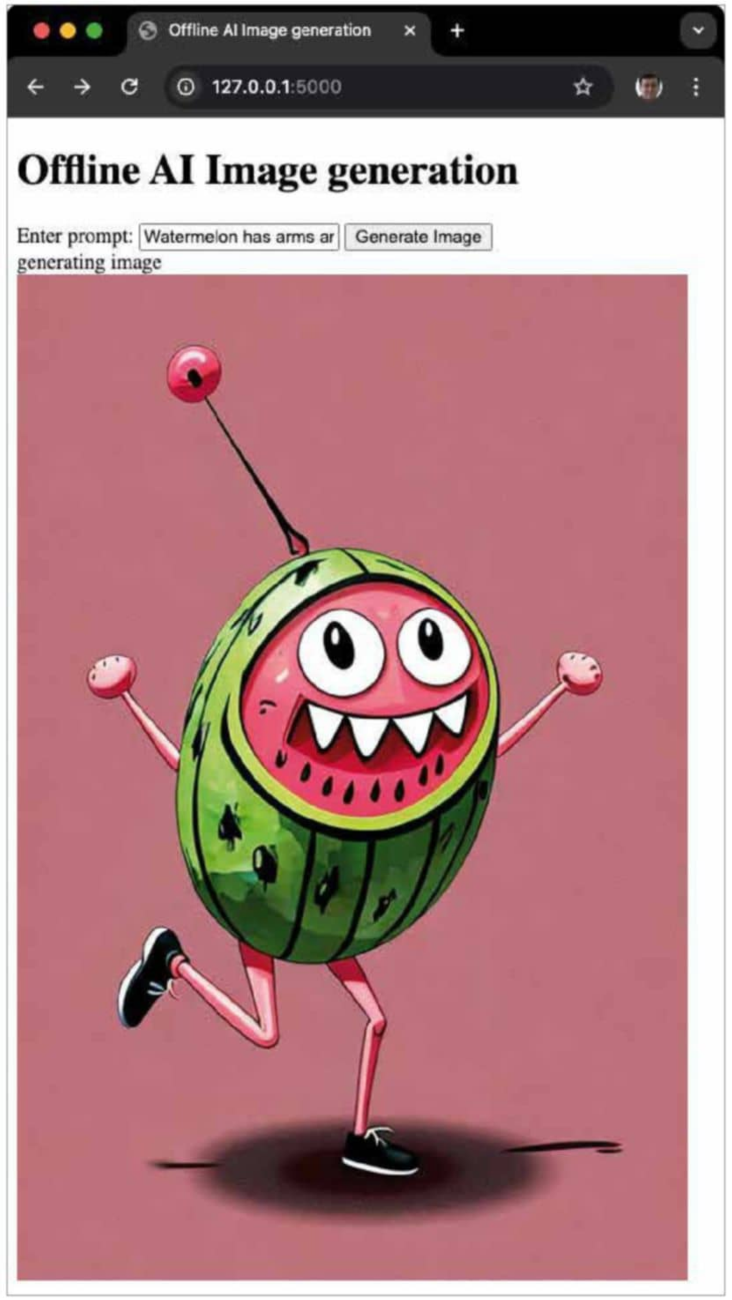


Offline AI Image generation

Enter prompt: A cat wearing a Batman

generating image

Figure 8: Purr-man to the rescue! Protecting NYC from evil, one catnap at a time.



Offline AI Image generation

Enter prompt: Watermelon has arms ar

generating image

Figure 9: Melon-choly no more!

Tell me what real-world tasks you'd like to see AI solve for you. I'd like to keep a developer focus, and ideally target solutions that don't require subscriptions or swiping credit cards.

Hey, PowerPoint! Where is my AI copilot for memes?

More next time. Until then, happy coding!

Sahil Malik

SPONSORED SIDEBAR:

CODE Is Hiring!

CODE Staffing is accepting resumes for various open positions ranging from junior to senior roles.

We have multiple openings and will consider candidates who seek full-time employment or contracting opportunities. For more information: www.codestaffing.com.

CODE Magazine Presents: The State of AI Mini Conference Tour

CODE has recently started a new series of in-person events focusing on the topic of artificial intelligence and its practical uses in business scenarios. We call this our **State of AI: Generating Real Business Value with AI** tour and they're essentially a one-day mini conference with five sessions and two panels. In June 2024 we held events in Houston and Dallas, and by the time you read this we'll present in a few other US cities, with additional locations later in 2024 or early 2025.

You can request an invitation to attend an in-person State of AI event. Because (so far) we are presenting these events in meeting space provided by Microsoft in the local area, we must manage capacity closely. The event features multiple speakers, including CODE Group's founder and Microsoft Regional Director, **Markus Egger, CODE Consulting's CEO, Mike Yeager, CODE Staffing's CEO, Yair Alan Griver, as well as several others from our senior technical staff who are working on AI projects.**

The day's agenda begins with a keynote by Markus Egger, looking at the state of artificial intelligence and its use in business applications in a way that provides concrete value. (As Markus likes to say: "The time for AI to move from having a lot of potential to providing concrete value is now!") The day then moves on to take a close look at how AI projects fail followed by how to succeed with AI. After a catered lunch, we have a highly anticipated business-focused panel discussion titled **No Answer Begins with "It Depends."** This unique approach has proven to be extremely popular. Audience members can pre-submit questions or ask them on the spot. The panelists are challenged to provide concrete answers, as "It depends" is explicitly not allowed.

About Our Presenter
START HERE
CODE

r at the Ion District, Houston, TX

> Very insightful presentations by CODE on the practicality of AI in today's business world.
>
> *Chris Zuniga via LinkedIn*

After the business-focused panel and a break, one of CODE's senior software architects presents a deeper technical dive showing how developers might approach using AI in custom applications. The day finishes with a technical panel discussion. We deliberately organize sessions to start less technical in the morning, giving consideration to decision makers who attend, but then end on a more technical note to also provide information to those not afraid of a detailed technical discussion. In our first two events, a large part of the audience stayed beyond the final afternoon break, and the technical panel discussion and Q&A proved to be a very entertaining and informative session.

> As a developer who is exploring the use cases and implementations of AI, this was a very informative and innovative forum.
>
> *Dhinesh Kumar Ramaswami via LinkedIn*

Nevertheless, the overall day is designed to be valuable to non-technical managers and decision-makers, to provide solid information to people who must engage in AI projects or must decide about whether or not such projects even apply to their organization. The day also features breakfast, lunch, and several breaks, specifically designed to enable conversations between attendees and CODE speakers and amongst attendees.

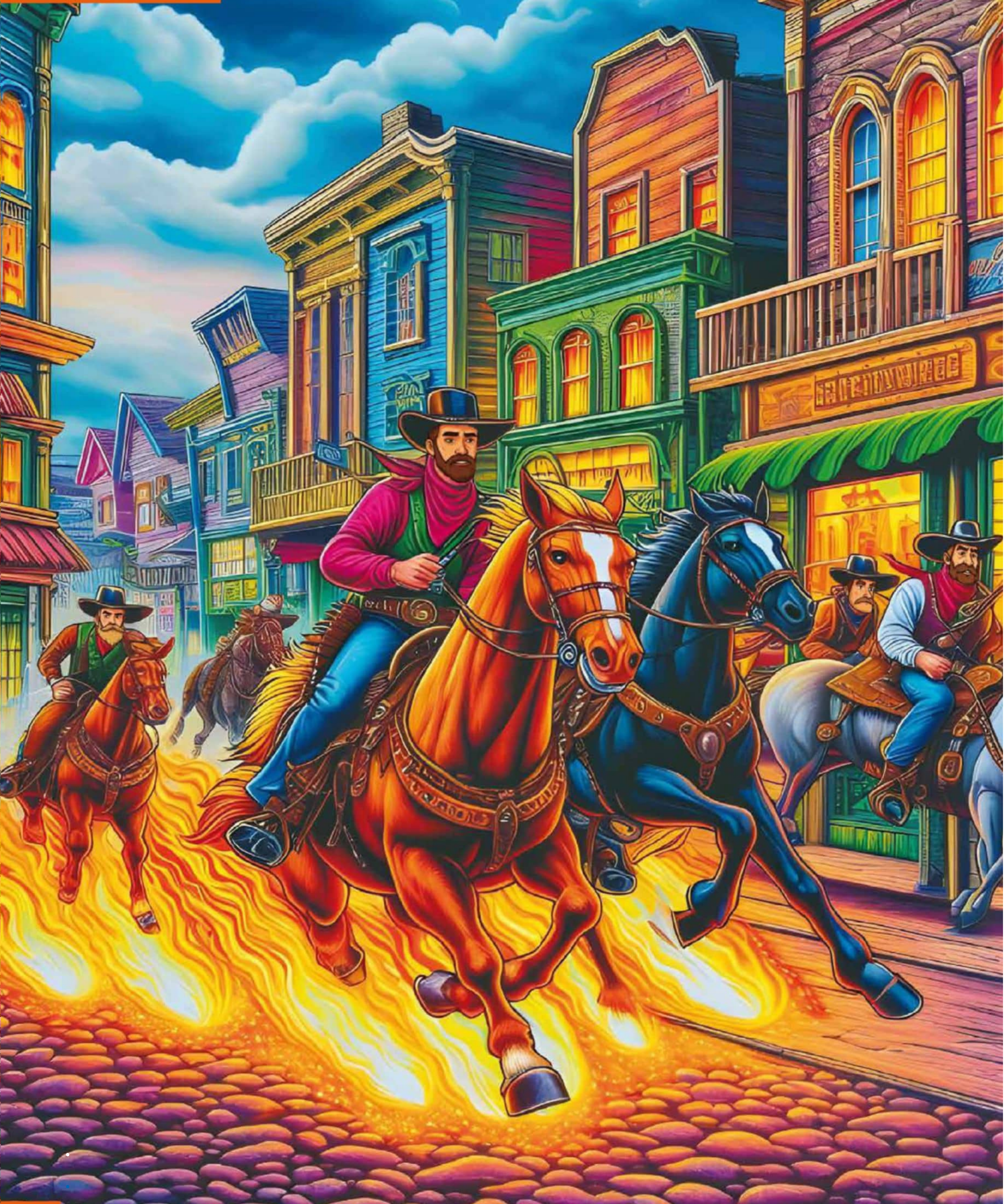
> Being in a room with like-minded AI advocates pursuing the augmentation of human intelligence was truly inspiring…
> It was fascinating to hear different perspectives on AI's potential.
>
> *Leopoldo Torres via LinkedIn*

> I attended the CODE Magazine State of AI event at the Microsoft Dallas, TX offices today. It was very informative. My biggest take away from this event is ...
> I can do this.
>
> *Itrel Monroe via LinkedIn*

Cannot Make It?

For people that cannot make it to an in-person State of AI event, but have an interest in how these topics apply to their company's specific scenarios or businesses, they can request an **AI Executive Briefing**, a service offered by CODE Consulting. AI Executive Briefings are offered in two "sizes." In our (free, online) short version (two hours or so) we provide a presentation to a company's senior managers, and we answer their more urgent questions. In our larger AI Executive Briefing, we have a more in-depth presentation and that leads to an actual pilot project and in-depth technical analysis (individually priced). Either version is a great way to get your management team, executives, or a board of directors up to speed with what AI can provide that is of concrete value for each business. Our AI Executive Briefings are a very popular service, which means we schedule them on a first-come, first-serve basis (and subject to available capacity).

Content paid for by codemag.com



Building Modern Web Applications Using Blazor ASP.NET Core

Blazor is a modern web framework from Microsoft that was included in .NET 5. It's used for building interactive web applications using C# and .NET and it's based on a flexible, modular component model that's well-suited for building applications with rich, interactive web user interfaces. It should be noted that you can still use JavaScript if you'd like to, i.e., you can invoke your JavaScript functions from C# and vice versa. This article will take a deep dive into Blazor and its components, and then demonstrate how to build modern web applications using it. It will also discuss the performance and security considerations, deployment using Docker and Kubernetes, and the best practices.

If you're to work with the code examples discussed in this article, you need the following installed in your system:

- Visual Studio 2022
- .NET 9.0
- ASP.NET 9.0 Runtime

If you don't already have Visual Studio 2022 installed on your computer, you can download it from here: https://visualstudio.microsoft.com/downloads/.

•

At the end of this journey, you'll be able to build high-performance, scalable, and secure Blazor applications in ASP.NET Core 9 and Visual Studio 2022.

Understanding the Problem

While building web applications, you would typically need server- and client-side components. To create the server-side components, you might want to use C#, Java, etc. On the other hand, when building the client-side components, you might typically want to use Angular, React, etc. You need to learn and master two different types of technologies, one for the server side and one for the client side. This makes it difficult to maintain the codebase over time. See **Figure 1** to understand a technology stack of a typical web application that doesn't use Blazor.

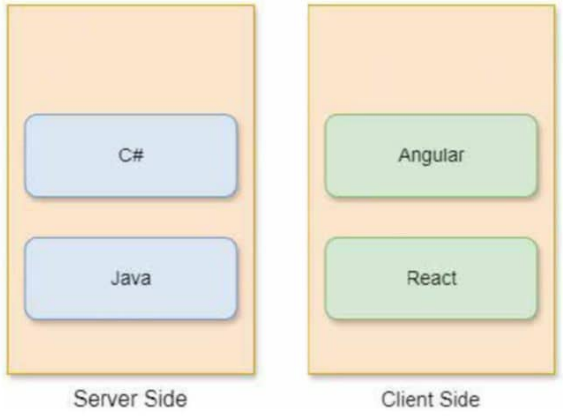


Figure 1: The technology stack in a typical web application

Blazor is a web framework that allows web developers to use C# in lieu of JavaScript to create modern web applications with reusable components that can be executed on both the client-side and the server-side to provide optimal web solutions. The Blazor framework is part of the ASP.NET Core framework and helps in streamlining the web development process for .NET and C# developers targeting web applications.

An Introduction to Blazor

Blazor, an open-source framework from Microsoft, empowers developers to create interactive web applications using C# and .NET. Blazor employs WebAssembly, a web standard that permits browser-based code execution from languages other than JavaScript, creating a fully featured, high-performance client-side development environment using C#. By using .NET, Blazor empowers developers to build full-stack applications encompassing both client and server components. Such flexibility permits developers to choose the hosting model best suited to their application's needs, including client-side rendering for interactive interfaces and server-side rendering for improved performance and scalability.

Blazor provides a modern, efficient, and versatile approach to web application development. It allows developers to build interactive, high-performing applications using C# and the .NET and .NET Core frameworks. By using its features, tools, and best practices, developers can streamline development, improve application functionality, and create engaging user experiences across diverse platforms.

Key Features of Blazor

Here's a quick look at the key features of Blazor.

Using C# in Lieu of JavaScript Throughout

When using Blazor, developers no longer need to leverage multiple development languages, libraries, and tools when building their applications. Instead, they can use C# throughout all application layers, both at the server side and the client-side, thereby eliminating the need to learn and use JavaScript. Consequently, this reduces the development effort, enables you to use the same language for building your client- and server-side components, thereby promoting code reuse across different platforms.

Component-Based Architecture

Blazor follows a component-based architecture, facilitating code reusability, separation of concerns, modular design, easy maintenance, and enabling you to build ap-

Joydip Kanjilal
joydipkanjilal@yahoo.com

Joydip Kanjilal is an MVP (2007-2012), software architect, author, and speaker with more than 20 years of experience. He has more than 16 years of experience in Microsoft .NET and its related technologies. Joydip has authored eight books, more than 500 articles, and has reviewed more than a dozen books.

plications in a structured manner. These components take the form of self-contained, reusable units of UI and logic written in razor syntax, which combines HTML, CSS, and C#. These components specify your application's logic and structure, making multilevel web development possible. In this approach, the source code is divided into smaller logical pieces that can be used again in the future while making the applications responsive and interactive.

Authentication and Authorization

Blazor provides support for robust security and data protection to thwart malicious attackers. It comes with built-in support for authentication and authorization and easy integration with OAuth providers, IdentityServer, and Azure AD. Note that Blazor takes advantage of the ASP.NET security framework to establish a user's identity. In Blazor Server apps, the AuthenticationStateProvider service uses the HttpContext.User API to retrieve authentication state data.

> Identity Server is an open-source framework for implementing identity and access control in your .NET applications. It implements OpenID Connect (OIDC) and OAuth 2.0 standards, integrates with ASP.NET Core Identity framework, and provides a common way to authenticate requests in ASP.NET Core applications.

Integration with the .NET Ecosystem

Blazor integrates nicely with the .NET ecosystem, including ASP.NET Core, Entity Framework Core, etc. You can easily share code between the server and client components, thereby enhancing maintainability and reducing code redundancy. Additionally, you can call your JavaScript functions easily from Blazor and integrate Blazor with JavaScript frameworks and libraries.

Support for Multiple Hosting Models

Blazor supports both client-side and server-side hosting models, providing flexibility and performance optimization based on project requirements. To achieve this, it supports multiple hosting models, such as the following:

- **Blazor WebAssembly (WASM):** These applications can be executed entirely on the client-side in the browser using WebAssembly.
- **Blazor Server:** These are applications that run on the server and can transmit UI updates to the client components over a SignalR connection.
- **Blazor Hybrid:** These applications combine the best of both worlds, i.e., they blend Blazor with .NET MAUI for building native desktop and mobile apps.

Platform Independence

Applications built in Blazor are platform independent. For example, you can run Blazor WebAssembly apps in modern browsers without using plug-ins. Moreover, you can execute Blazor Hybrid apps on Windows, macOS, iOS, and Android platforms using .NET MAUI. Additionally, you can host your Blazor Server applications on any platform that provides support for ASP.NET Core.

Render Modes

Blazor leverages render modes to determine the hosting model to be used, whether the application should be rendered at the server or the client, and whether it's interactive or non-interactive. Blazor supports the following types of render modes:

- Static server
- Interactive server
- Interactive WebAssembly
- Interactive auto

How Does Blazor Work?

Blazor is a Microsoft framework that allows executing applications built using C# in a web browser without using any plug-ins. You can use Blazor to build modern-day applications using C# as a full-stack development tool. Applications developed using Blazor run inside the context of your web browser. Once you compile such an application, one or more files are loaded into the browser and executed. Unlike ASP.NET Core, you don't need a specific back-end component to run your Blazor application. Blazor applications can access web services using HTTP REST APIs.

A Blazor application is composed of optional reusable components containing a C#, HTML, and CSS conglomerate that can run both on the server and client sides. These components define the structure and behavior of the user interface. These are .NET C# classes enclosed in .NET assemblies supporting event handling logic and user events that can be reused and also be made available as Razor class libraries or NuGet packages. Components handle user interaction using events such as button clicks, which trigger updates to the state of the components.

Blazor WebAssembly

Blazor WebAssembly (WASM) is a single-page application framework for building cutting-edge client-side web applications based on .NET that are compatible with all web browsers. With Blazor WASM, the entire application, from the application logic to UI elements, its dependencies, and the .NET Core runtime, is loaded in the web browser. Anytime you launch the web application or any web page, the code responsible for the client-side logic and all its dependencies is also fetched. The source code for .NET WASM and its dependencies, like C# and Razor files, are compiled into .NET assemblies, which are then transferred to your web browser and executed there. It's preserved in bytecode format for fast download and execution and allows interaction with the browser through JavaScript, using a feature known as JavaScript Interop. **Figure 2** shows the components of a typical Blazor Web Assembly application.

Blazor Server

In the Blazor server hosting model, components run on the server inside an ASP.NET Core application. Using the SignalR connection and the WebSockets protocol, you manage UI changes or updates, events, and JavaScript calls. The Blazor application is hosted on the server, and all changes or events triggered on the application's browser are simplified and communicated to the server using SignalR. Although the user interface is rendered to the web browser, the UI updates and event handling are performed on the server side. Although this is analogous to traditional web applications, unlike a traditional web application, the client and the server communicate over a SignalR connection, as shown in **Figure 3**.

Blazor Hybrid

The Blazor Hybrid model enables developers to create native applications for mobile or desktop using a hybrid model in which Razor components are executed directly in the .NET process instead of through WebAssembly. This approach doesn't require web assembly to execute your application. You can take advantage of .NET native technologies, such as MAUI and WPF, together with Razor components to create your Blazor Hybrid applications. **Figure 4** shows the components of a Blazor Hybrid application.

Use Cases

Here are a few use cases for Blazor applications:

- Single-page applications (SPAs)
- Progressive web apps (PWAs)
- Cross-platform desktop applications
- Real-time applications
- Line-of-business (LOB) applications

What Are Progressive Web Applications?

Progressive Web Applications (PWAs) are web applications that provide users with a rich experience regardless of the platform on which they execute.

The following are the features of PWAs:

- **Progressive enhancement:** PWAs can work on any device or platform, enabling an intuitive user experience on desktops and mobile devices.
- **Responsive design:** PWAs provide an enriched user experience by following responsive web design principles across devices and platforms.
- **Offline support:** PWAs can work offline or in areas with limited connectivity so that you can access the application even when there's no connectivity.

Here are the key benefits of PWAs:

- Cross-platform compatibility
- Offline Accessibility
- Enhanced performance
- Cost-effective

There are certain downsides to using PWAs:

- Security concerns
- Limited native support
- Limited user engagement
- Limited discoverability
- Limited browser support

In the application you'll create later in this article, you won't use PWAs for simplicity and brevity.

Create a New Blazor Web Assembly Project in .NET 9 and Visual Studio 2022

You can create a project in Visual Studio 2022 in several ways, such as from the Visual Studio 2022 Developer Command Prompt or by launching the Visual Studio 2022 IDE. When you launch Visual Studio 2022, you'll see the Start window. You can choose **Continue without code** to launch the main screen of the Visual Studio 2022 IDE.

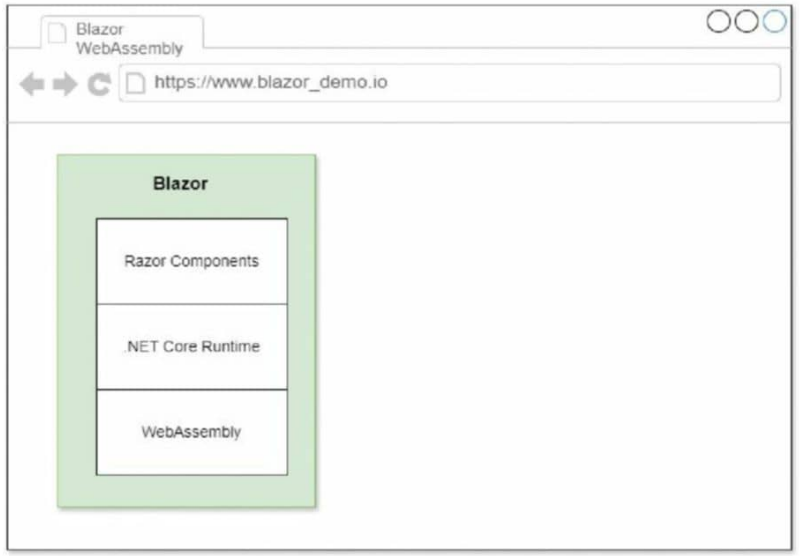


Figure 2: The components of a Blazor Web Assembly application

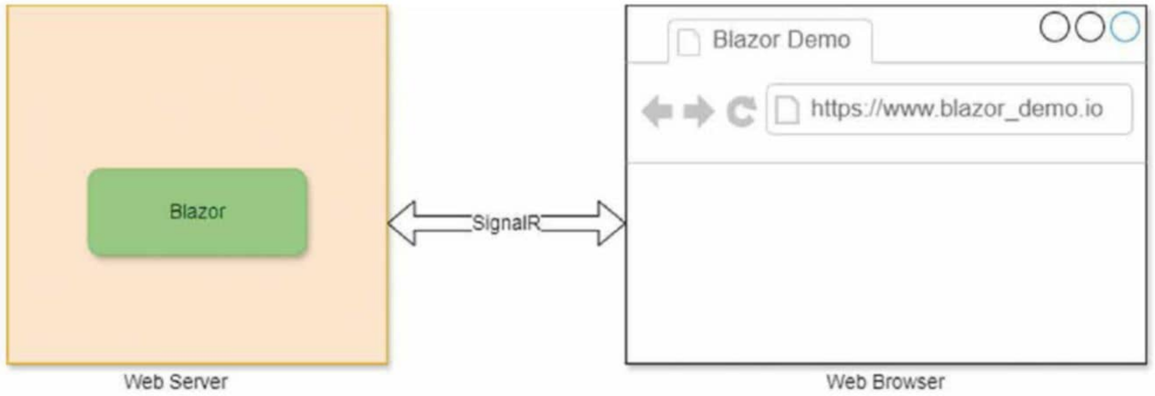


Figure 3: In a Blazor Server app, the client and the server applications communicate using SignalR

Now that you know the basics, let's start setting up the project. To create a new ASP.NET Core 8 Project in Visual Studio 2022:

1. Start the Visual Studio 2022 IDE.
2. In the **Create a new project** window, select **Blazor Web App** and click Next to move on.
3. Specify the project name as **Blazor_WebAssembly_Demo** and the path where it should be created in the **Configure your new project** window.

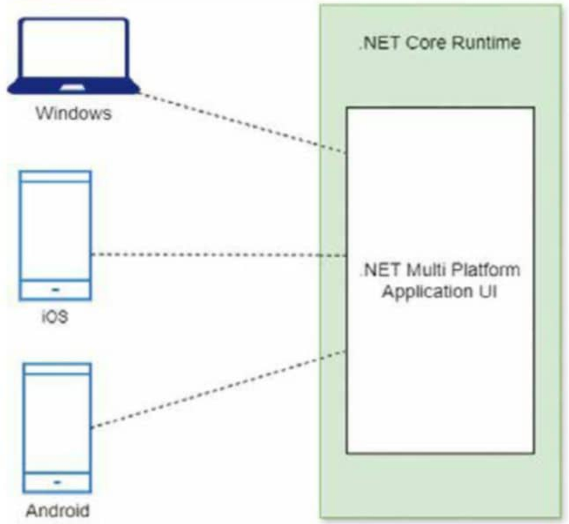
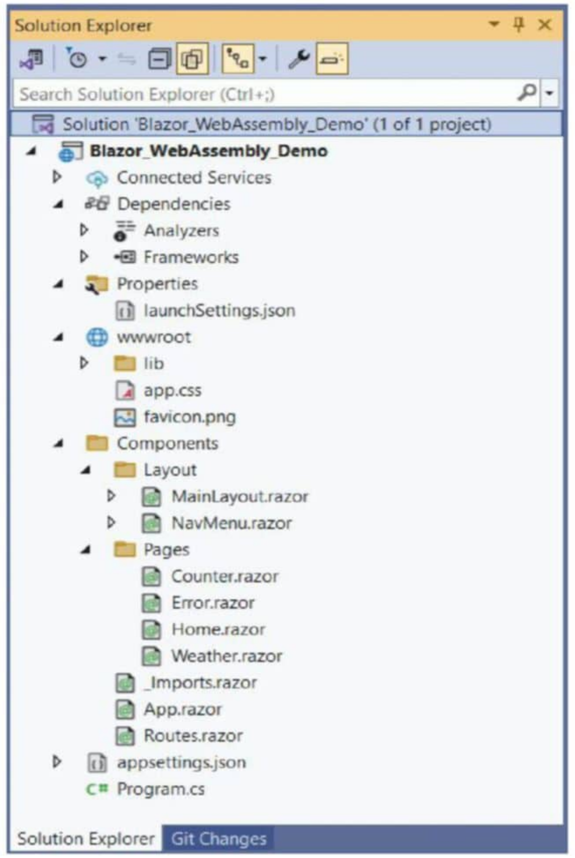


Figure 4: Components of a Blazor Hybrid application



Solution 'Blazor_WebAssembly_Demo' (1 of 1 project)
Blazor_WebAssembly_Demo
Connected Services
Dependencies
Analyzers
Frameworks
Properties
launchSettings.json
wwwroot
lib
app.css
favicon.png
Components
Layout
MainLayout.razor
NavMenu.razor
Pages
Counter.razor
Error.razor
Home.razor
Weather.razor
_Imports.razor
App.razor
Routes.razor
appsettings.json
Program.cs

Figure 5: The solution structure of a Blazor Web Assembly application

4. If you want the solution file and project to be created in the same directory, you can optionally check the **Place solution and project in the same directory** checkbox. Click Next to move on.
5. In the next screen, specify the target framework and authentication type as well. Ensure that the **Configure for HTTPS**, and **Do not use top-level statements** checkboxes are unchecked because you won't use any of these in this example.
6. Next, specify the **Interactive render mode** and **Interactivity location**
7. You should ensure that the **Include sample pages** checkbox is checked if you would like to have sample pages added to your project
8. Click **Create** to complete the process.

A new Blazor Web App project is created. **Figure 5** shows the default solution structure.

When you execute the application, the Home page will be displayed in the web browser, as shown in **Figure 6**.

Integrating Blazor WebAssembly into an Existing ASP.NET Core Web Application

In this section, you'll examine how you can integrate a Blazor WebAssembly application with an existing ASP.NET Core Web Application.

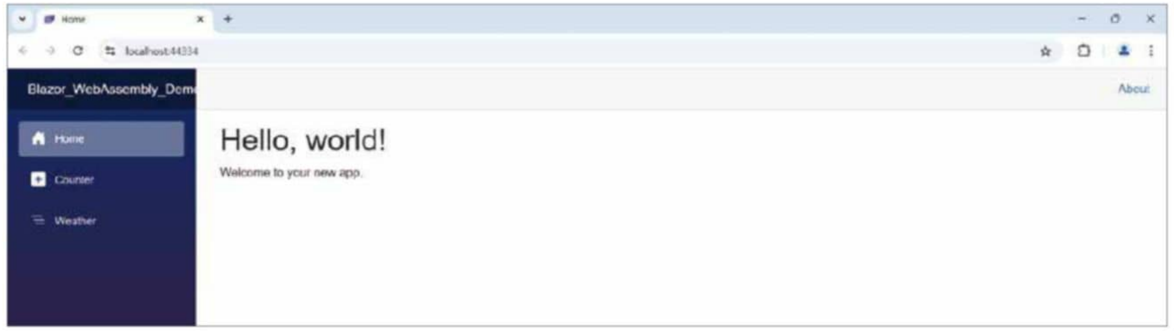
First, create an ASP.NET Core 9 application in Visual Studio by following the steps outlined in the next section.

Create a New ASP.NET Core 9 Project in Visual Studio 2022

You can create a project in Visual Studio 2022 in several ways, such as from the Visual Studio 2022 Developer Command Prompt or by launching the Visual Studio 2022 IDE. When you launch Visual Studio 2022, you'll see the Start window. You can choose **Continue without code** to launch the main screen of the Visual Studio 2022 IDE.

Now that you know the basics, let's start setting up the project. To create a new ASP.NET Core 8 Project in Visual Studio 2022:

1. Start the Visual Studio 2022 IDE.
2. In the **Create a new project** window, select **ASP.NET Core Web API** and click Next to move on.
3. Specify the project name as **ASPNETCoreBlazor** and the path where it should be created in the **Configure your new project** window.
4. If you want the solution file and project to be created in the same directory, you can optionally check the **Place solution and project in the same directory** checkbox. Click Next to move on.
5. In the next screen, specify the target framework and authentication type as well. Ensure that the **Configure for HTTPS**, **Enable Docker Support**, **Do not use top-level statements**, and the **Enable OpenAPI support** checkboxes are unchecked because you won't use any of these in this example.
6. Remember to leave the **Use controllers** checkbox checked because you won't use minimal API in this example.
7. Click Create to complete the process.



Hello, world!

Welcome to your new app.

Figure 6: A Blazor Web Assembly application in execution

A new ASP.NET Core Web API project is created. You'll use this project to implement the CQRS pattern in ASP.NET Core and C#.

Next, create a new Blazor WebAssembly application in the same solution using the steps outlined in an earlier section and follow the steps given below to integrate your Blazor WebAssembly into the ASP.NET Core Web project.

1. Right-click on the Solution Explorer and select the Add New Project option to create a new project into your solution.
2. Next, search for Blazor WebAssembly and select Blazor WebAssembly App from the search results found.
3. Configure the Blazor WebAssembly App using the **Configure your new project** dialog.
4. In the Program.cs file, comment out this line: **builder.RootComponents.Add<App>("#app");**.
5. To enable your ASP.NET Core Web application to use the Blazor Web Assembly components, add the Blazor application as a project reference using the Reference Manager dialog window.
6. Search for the NuGet project Microsoft.AspNetCore.Components.WebAssembly.Server in the ASP.NET Core Web application and add install it.
7. Add the following two lines in the Program.cs file:

```
app.UseBlazorFrameworkFiles();
app.MapFallbackToFile("index.html");
```

1. Add the following script reference as well:

```
<script src="_framework/blazor.webassembly.js">
</script>
```

That's all you need to do!

In the next section, you'll create a new Blazor Server application in Visual Studio 2022.

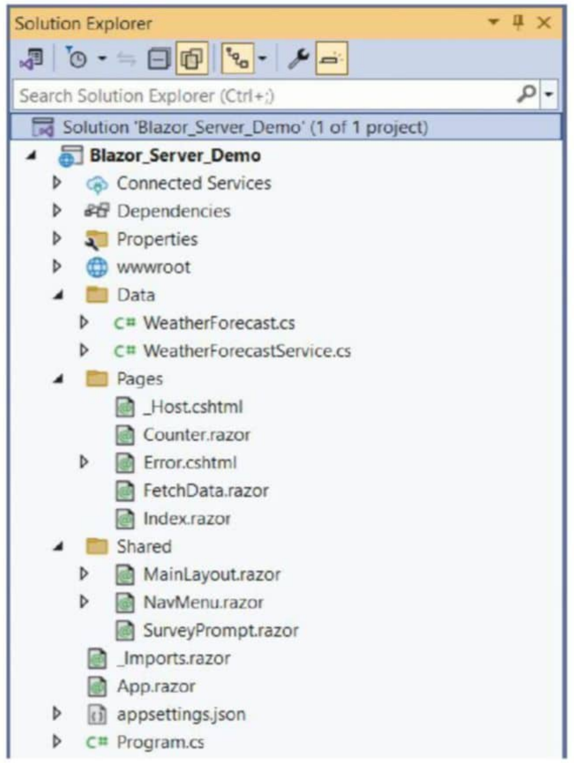
Create a New Blazor Server Application in .NET 9 and Visual Studio 2022

You can create a project in Visual Studio 2022 in several ways, such as from the Visual Studio 2022 Developer Command Prompt or by launching the Visual Studio 2022 IDE. When you launch Visual Studio 2022, you'll see the Start window. You can choose **Continue without code** to launch the main screen of the Visual Studio 2022 IDE.

Now that you know the basics, let's start setting up the project. To create a new ASP.NET Core 8 Project in Visual Studio 2022:

1. Start the Visual Studio 2022 IDE.
2. In the **Create a new project** window, select **Blazor Server App** and click Next to move on.
3. Specify the project name as **Blazor_Server_Demo** and the path where it should be created in the **Configure your new project** window.
4. In the next screen, specify the target framework and authentication type as well. Ensure that the **Configure for HTTPS** and **Do not use top-level statements** checkboxes are unchecked because you won't use any of these in this example.
5. If you want the solution file and project to be created in the same directory, you can optionally check the **Place solution and project in the same directory** checkbox. Click Next to move on.
6. Click Create to complete the process.

A new Blazor Server App project is created. **Figure 7** shows the default solution structure of the Blazor Server App you just created.



Solution 'Blazor_Server_Demo' (1 of 1 project)
Blazor_Server_Demo
Connected Services
Dependencies
Properties
wwwroot
Data
WeatherForecast.cs
WeatherForecastService.cs
Pages
_Host.cshtml
Counter.razor
Error.cshtml
FetchData.razor
Index.razor
Shared
MainLayout.razor
NavMenu.razor
SurveyPrompt.razor
_Imports.razor
App.razor
appsettings.json
Program.cs

Figure 7: The solution structure of a Blazor Server application

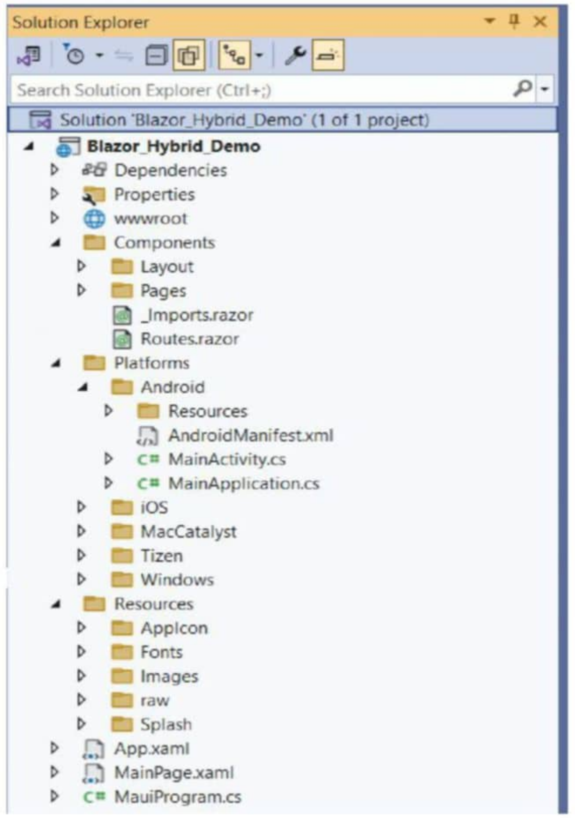
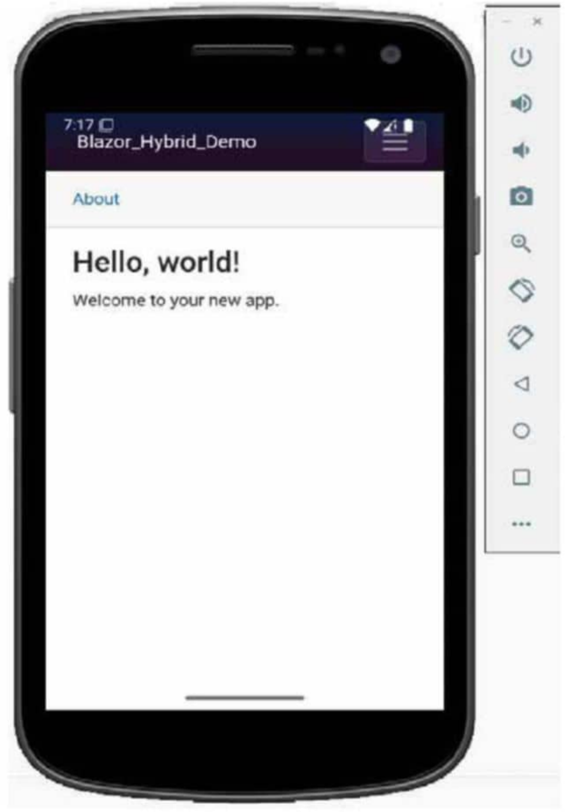

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Blazor_Hybrid_Demo' (1 of 1 project)
Blazor_Hybrid_Demo
Dependencies
Properties
wwwroot
Components
Layout
Pages
_Imports.razor
Routes.razor
Platforms
Android
Resources
AndroidManifest.xml
MainActivity.cs
MainApplication.cs
iOS
MacCatalyst
Tizen
Windows
Resources
AppIcon
Fonts
Images
raw
Splash
App.xaml
MainPage.xaml
MauiProgram.cs

Figure 8: The solution structure of a Blazor Hybrid application


7:17
Blazor_Hybrid_Demo
About

Hello, world!

Welcome to your new app.

Figure 9: A Blazor Hybrid application running in an Android emulator

Create a New Blazor Hybrid Application in .NET 9 and Visual Studio 2022

Set up a new Blazor Hybrid Application project in Visual Studio 2022 by following the steps outlined below:

1. Start the Visual Studio 2022 IDE.
2. In the **Create a new project** window, select **.NET MAUI Blazor Hybrid App** and click Next to move on.
3. Specify the project name as **Blazor_Hybrid_Demo** and the path where it should be created in the **Configure your new project** window.
4. In the next screen, specify the target framework and authentication type as well. Ensure that the **Configure for HTTPS** and **Do not use top-level statements** checkboxes are unchecked because you won't use any of these in this example.
5. If you want the solution file and project to be created in the same directory, you can optionally check the **Place solution and project in the same directory** checkbox. Click Next to move on.
6. Click Create to complete the process.

Figure 8 shows what the solution structure of the Blazor Hybrid Application looks like.

Listing 1: The Database Script

```
 -- Supplier table

CREATE TABLE Supplier (

    Supplier_Id UniqueIdentifier PRIMARY KEY,
    Supplier_Name VARCHAR(255) NOT NULL,
    Supplier_Address VARCHAR(255) NOT NULL,
    Supplier_Contact VARCHAR(255),
    Supplier_Phone VARCHAR(20),
    Supplier_Email VARCHAR(20)
);

-- Product table

CREATE TABLE Product (

    Product_Id UniqueIdentifier PRIMARY KEY,
    Product_Name VARCHAR(255) NOT NULL,
    Product_Description TEXT,
    Unit_Price DECIMAL(10, 2) NOT NULL,
    Product_Quantity INT NOT NULL
);

-- Order table

CREATE TABLE [Order] (

    Order_Id UniqueIdentifier PRIMARY KEY,
    Product_Id UniqueIdentifier NOT NULL,
    Supplier_Id UniqueIdentifier NOT NULL,
    Order_date DATE NOT NULL,
    Order_Quantity INT NOT NULL,
    FOREIGN KEY (Product_Id) REFERENCES Product(Product_Id),
    FOREIGN KEY (Supplier_Id) REFERENCES Supplier(Supplier_Id)
);

-- Shipment table

CREATE TABLE Shipment (

    Shipment_Id UniqueIdentifier PRIMARY KEY,
    Order_Id UniqueIdentifier NOT NULL,
    Shipment_Date DATE NOT NULL,
    Estimated_Arrival_Date DATE NOT NULL,
    Actual_Arrival_Date DATE,
    FOREIGN KEY (Order_Id) REFERENCES [Order](Order_Id)
);
```

You can execute this application in Windows (default) or your Android and iOS devices. To run this application on an Android device, follow these steps:

1. From the menu in your Visual Studio IDE, select Tools > Android > Android Device Manager.
2. When the Android Device Manager is launched, create a new Android device to run your application.
3. Once the Android Device has been created, click Start.
4. If you've got multiple emulators created for your Android device, select your desired emulator and then click F5 to run the application.

Figure 9 shows the Blazor Hybrid application running in an Android emulator.

Implement a Supply Chain Management Application

Microservices architecture is a structural approach that organizes an application as a collection of small independent services modeled around a business domain that can communicate with one another, if need be. In this section, you'll implement a Supply Chain Management Application using ASP.NET Core and Blazor. To do this, follow these steps:

1. Create the Supply Chain Management System database
2. Build the Solution Structure
3. Create the Product Microservice
   a. Create the Product model
   b. Create the Product Data Context
   c. Seed the Product database table
   d. Create the Product Repository
   e. Create the Product Controller
6. Create the Supplier Microservice
   a. Create the Supplier model
   a. Create the Supplier Data Context
   a. Seed the Supplier database table
   a. Create the Supplier Repository
   a. Create the Supplier Controller
2. Install Entity Framework Core
3. Configure the application
   a. Specify the database connection string
2. Execute the application

Create the Supply Chain Management System Database

Create a new database called SupplyChainManagementSystem using the following script:

```
Create database SupplyChainManagementSystem
```

Next, create the Product, Order, Supplier, and the Shipment database tables inside the SupplyChainManagementSystem database using the script given in **Listing 1**.

Figure 10 demonstrates the database diagram of the ShoppingCartSystem database.

In this next section, you'll create a new Blazor Server Application named SupplyChainManagementSystem by following the steps outlined earlier.

The Solution Structure

As evident from the database design, the SupplyChainManagementSystem application is comprised of the Supplier, Product, Shipment, and Order microservices. Each of these microservices correspond to a module. There are four modules in this application.

For the sake of simplicity, you'll create only two of them: **Supplier** and **Product. Figure 11** shows what the solution structure looks like.

There are several files and folders in the generated project that you can observe when you look at the solution structure:

- **Pages:** This solution folder contains Razor components or Blazor UI building blocks.

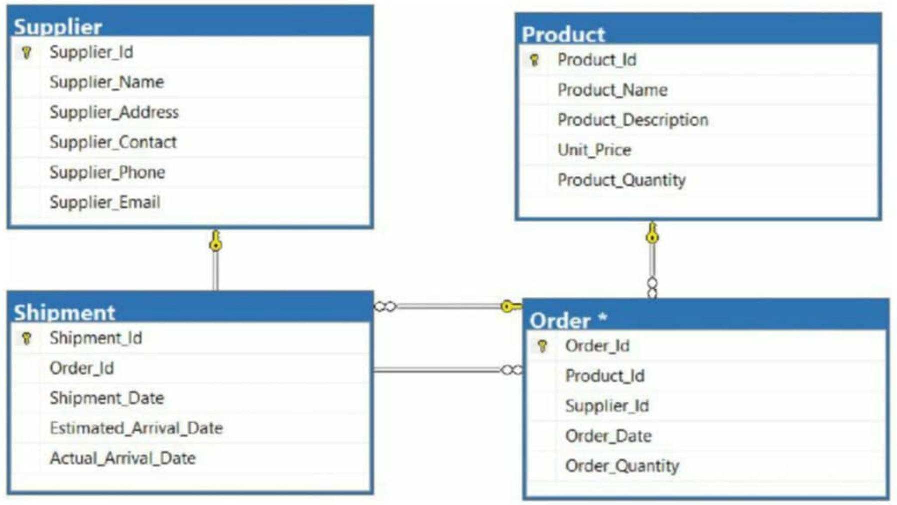

Supplier
Supplier_Id
Supplier_Name
Supplier_Address
Supplier_Contact
Supplier_Phone
Supplier_Email
Product
Product_Id
Product_Name
Product_Description
Unit_Price
Product_Quantity
Shipment
Shipment_Id
Order_Id
Shipment_Date
Estimated_Arrival_Date
Actual_Arrival_Date
Order *
Order_Id
Product_Id
Supplier_Id
Order_Date
Order_Quantity

Figure 10: The database design of the SupplyChainManagementSystem database

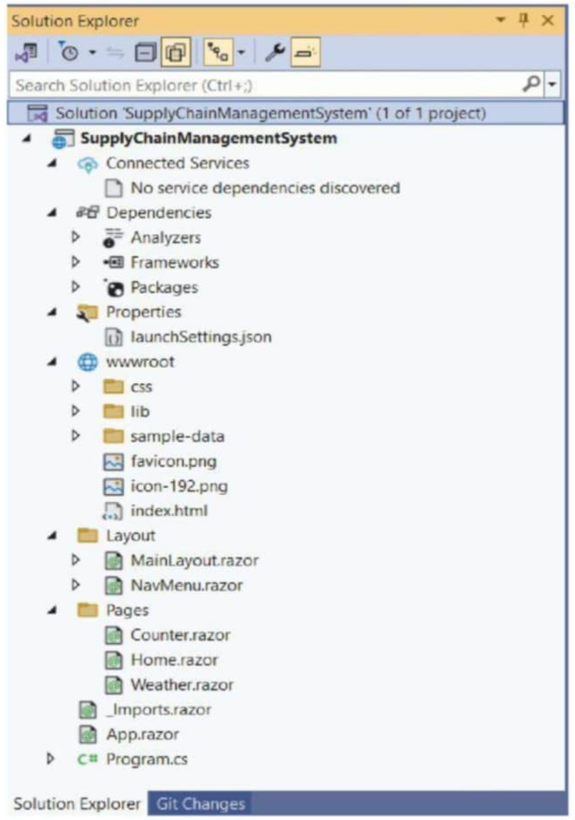

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SupplyChainManagementSystem' (1 of 1 project)
SupplyChainManagementSystem
Connected Services
No service dependencies discovered
Dependencies
Analyzers
Frameworks
Packages
Properties
launchSettings.json
wwwroot
css
lib
sample-data
favicon.png
icon-192.png
index.html
Layout
MainLayout.razor
NavMenu.razor
Pages
Counter.razor
Home.razor
Weather.razor
_Imports.razor
App.razor
Program.cs
Solution Explorer
Git Changes

Figure 11: The solution structure of our Supply Chain Management System

Microsoft Fabric
COMMUNITY CONFERENCE

March 31 - April 2, 2025 | WORKSHOPS: March 29, 30 & April 3, 2025

MGM GRAND, *Las Vegas, NV*

ARUN ULAG

Corporate Vice President, Azure Data, Microsoft

AMIR NETZ

Technical Fellow and CTO of Microsoft Fabric, Microsoft

KIM MANIS

Vice President of Product, Microsoft Fabric & Power BI, Microsoft

ADAM SAXTON

Principal Program Manager, Microsoft

PATRICK LEBLANC

Principal Program Manager, Microsoft

JUSTYNA LUCZNIK

Partner Group Product Manager, Microsoft Fabric Data Engineering & Data Science, Microsoft

PRIYA SATHY

Partner Director, PM, Data Warehousing, Microsoft

SHIREESH THOTA

Corporate Vice President, Azure Databases, Microsoft

WANGUI MCKELVEY

General Manager, Azure Data Analytics, Microsoft

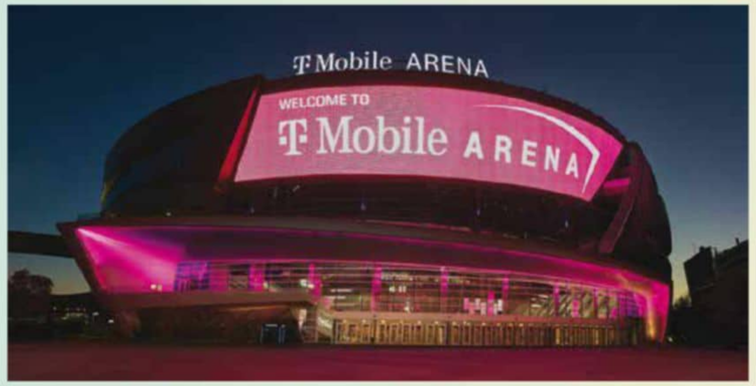
WEE HYONG TOK

Partner Director, Products, Microsoft

Register Today with Discount Code: CODEAD150

(Discount Is for New Registrations Only. Not Retroactive.)

FabricConf.com   @MSFabricConf   info@FabricConf.com



Don't miss the Opening Keynote held at the T-Mobile Arena!

New this #FABCON: *Fabric Databases*

Your favorite data experts will be front and center to share insights, demos, and deep-dives on SQL, Cosmos DB, PostgreSQL, and more.

RIE MERRITT

Principal PM Manager, Influencers, Events, & Community, Azure Data, Microsoft

BOB WARD

Principal PM Architect, Azure Data, Microsoft

MUAZMA ZAHID

Principal Group PM Manager, Microsoft

ERIN STELLATO

Principal Program Manager, Microsoft

DAVIDE MAURI

Principal Product Manager, Microsoft

Optional Full-Day Pre- and Post-Conference Workshops

Saturday, March 29, 2025

- Building an End-to-End Lakehouse Solution (Hands-on)
- End-to-End Data Engineering: Modern Data Warehousing on Microsoft Fabric (Hands-on)
- Fabric SQL Database Data Engineer in a Day (Hands-on)
- Govern Your Data on Fabric and Beyond: A Pre-Conference Exploration
- Make Better Decisions Faster with Real-Time Intelligence (Hands-on)
- Mastering Fabric Administration
- Microsoft Fabric for the Power BI Professional

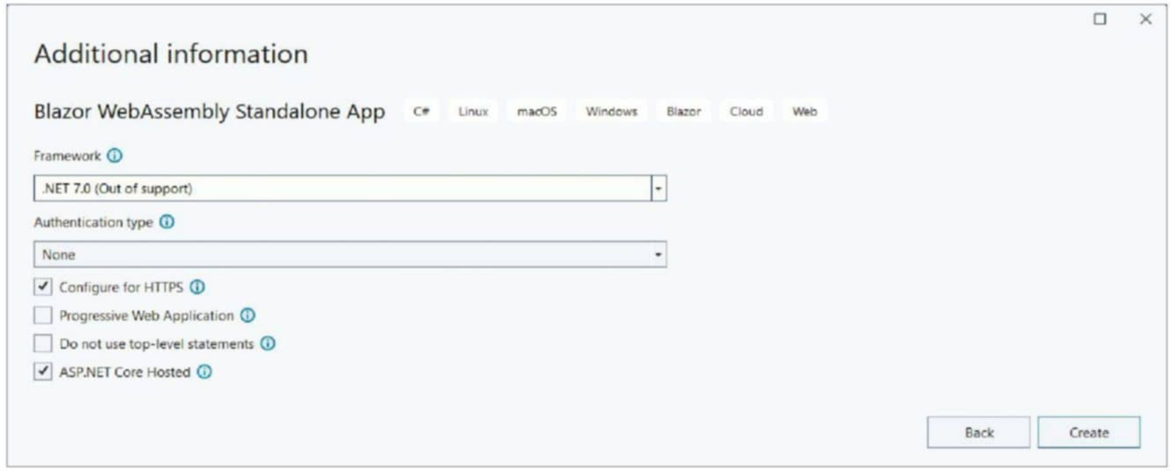
Sunday, March 30, 2025

- A Comprehensive Guide to Direct Lake for the Pro Data Modeller (Hands-on)
- Building AI Applications with SQL: Ground to Cloud to Fabric
- Data Factory: Learn What's New and Get from Zero to Hero in a Day
- Harnessing Apache Spark for Next-Gen Analytics in Microsoft Fabric (Hands-on)
- Professional Software Development on the Fabric Platform
- The Complete Guide to Certifications for a Microsoft Fabric Professional

Thursday, April 3, 2025

- Build Durable Multi-Agent Applications in C# or Python Using Azure Cosmos DB (Hands-on)
- Fabric Analyst in a Day
- Getting Started with AI & Data Science in Fabric
- Learn to Develop with Azure AI Services in Microsoft Fabric by Creating DP-700 Exam Study Materials
- Navigating End-to-End Power BI Development in the Cloud
- SQL Projects and Source Control Integration for SQL in Fabric, Azure SQL, and SQL Server
- What's New in Microsoft Purview?

JOIN US IN VEGAS!



Blazor WebAssembly Standalone App

Framework: .NET 7.0 (Out of support)

Authentication type: None

☑ Configure for HTTPS
☐ Progressive Web Application
☐ Do not use top-level statements
☑ ASP.NET Core Hosted

Figure 12: Creating a standalone Blazor WebAssembly App in DotNet7

- **wwwroot:** This solution folder contains static files like images, CSS, JavaScript etc.
- **Program.cs:** This file marks the entry point for the application.
- **App.razor:** This file represents the root component that configures routing and layout for your application.
- **_Imports.razor:** This file represents a Shared Razor imports file for the entire application.

In the sections that follow, you'll create the related classes and interfaces pertaining to the application you'll be building here.

Reorganizing the Project

When you create a standalone Blazor WebAssembly application in .NET 9, you'll not see any option to host it in ASP.NET Core. Microsoft has removed this option from .NET 8 onward. If you use .NET 7, you'll be able to see an option to host your project in ASP.NET Core, as shown in **Figure 12.**

> The server project is where we usually write the back-end logic (API, services, etc.). The shared project typically contains libraries that are used across both the server and client projects. The client project is where you build your user interface components.

When you create a standalone Blazor WebAssembly application in .NET 7, you'll observe three projects created automatically for you. These are the Server/API project, the Client project, and a Shared project.

You can, however, create these projects in .NET 9 manually as a workaround. In this example, you'll create the following three projects in the solution:

- SupplyChainManagementSystem.Server
- SupplyChainManagementSystem.Client
- SupplyChainManagementSystem.Shared

Install Entity Framework Core

So far, so good. The next step is to install the necessary NuGet Package(s) for working with Entity Framework Core and SQL Server. To install these packages into your project, right-click on the solution and then select **Manage NuGet Packages for Solution….**

Once the window pops up, search for the NuGet packages to add to your project. To do this, type in Microsoft.EntityFrameworkCore, Microsoft.EntityFrameworkCore.Design, Microsoft.EntityFrameworkCore.Tools, and Microsoft.EntityFrameworkCore.SqlServer in the search box and install them one after the other. Alternatively, you can type the commands shown below at the NuGet Package Manager Command Prompt:

```
PM> Install-Package
Microsoft.EntityFrameworkCore

PM> Install-Package
Microsoft.EntityFrameworkCore.Design

PM> Install-Package
Microsoft.EntityFrameworkCore.Tools

PM> Install-Package
Microsoft.EntityFrameworkCore.SqlServer
```

You can also install these packages by executing the following commands at the Windows Shell:

```
dotnet add package
Microsoft.EntityFrameworkCore
dotnet add package
 Microsoft.EntityFrameworkCore.Design
dotnet add package
 Microsoft.EntityFrameworkCore.Tools
dotnet add package
 Microsoft.EntityFrameworkCore.
SqlServer
```

Install the NuGet packages in both the SupplyChainManagementSystem.Server and SupplyChainManagementSystem.Client (if you need to use EF Core components) projects.

Create the Product Microservice

In this example, you'll build the Product microservice application or the Product API. The product microservice application is composed of the following files:

- **Product.cs:** The product model that contains domain-specific data and (optionally) business logic
- **IProductRepository.cs:** The IProductRepository interface that contains the declaration of the operations supported by the product repository
- **ProductRepository.cs:** The product repository class that implements the members of the IProductRepository interface
- **ProductDbContext.cs:** The product data context used to perform CRUD operations for the Product table in the database
- **appsettings.json:** The application's settings file where you can configure the database connection string, logging metadata, etc.
- **Program.cs:** Any ASP.NET Core application contains a file where the startup code required by the application resides. This file is named Program.cs where the services required by your application are configured. You can specify dependency injection (DI), configuration, middleware, and much more information in this file.

Specify the Database Connection String

Your application requires a connection string to establish a connection to the database which, in turn, contains the necessary information about the database connection and any initialization parameters sent by a data provider to a data source. Typically, a connection string contains the name of the database to connect to, the instance name of the database server where the database resides, and some other settings pertaining to security of the database.

In ASP.NET Core, the application's settings are stored in a file known as appsettings.json. This file is created by default when you create a new ASP.NET Core project. You can take advantage of the ConnectionString property to retrieve or store the connection string for a database. You can specify the connection string in the appsettings.json file, as shown below:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    " DefaultConnection":
    "Write your connection string here."
  },
  "AllowedHosts": "*"
}
```

You'll use this connection string to enable the application to connect to the database in a section later in this article.

> Typically, you should access the database components in the server project only. If you're to implement authentication and authorization in your application, you can (optionally) do that in the client project.

Create the Model Classes

First off, create two solution folders named Models and DataAccess. The former will contain one or more model classes, and the latter will have the data context and repository interfaces and classes. It should be noted that you can always create multiple data context classes in the same project. If your data context class contains many entity references, it's a good practice to split the data context among multiple data context classes rather than having one large data context class.

Create a new class called Product in a file named Product.cs inside the Models folder and write the following code in there:

```
namespace SCMS.Product.Models
{
    public record Product
    {
        public Guid Product_Id
        { get; set; }

        public string Product_Name
        { get; set; } = default!;

        public string Product_Description
        { get; set; } = default!;

        public string Product_Category
        { get; set; } = default!;

        public decimal Product_Price
        { get; set; } = default!;

        public int Product_Quantity
        { get; set; } = default!;

        public DateTime Created_At
        { get; set; } = DateTime.Now;

        public DateTime Modified_At
        { get; set; } = DateTime.Now;
    }
}
```

In this implementation, you'll use only one model class: Product.

Create the Data Context

In Entity Framework Core (EF Core), a data context is a component used by an application to interact with the database and manage database connections, and to query and persist data in the database. Let's now create the data context class to enable the application to interact with the database to perform CRUD (Create, Read, Update, and Delete) operations.

To do this, create a new class named ProductDbContext that extends the DbContext class of EF Core and write the following code in there:

```
public class ProductDbContext: DbContext
{
    public DbSet <Product>
    Products { get; set; }

    protected override void OnConfiguring
    (DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring (optionsBuilder);
    }
}
```

In the preceding piece of code, the statement base.OnConfiguring(optionsBuilder) calls the OnConfiguring method of the base class of your ProductDbContext. Because the base class of the ProductDbContext class is DbContext, the call does nothing in particular.

Hosting Models in Blazor

Blazor offers a range of hosting models and gives flexibility to developers in terms of whether to host the application on the client-side or on the server-side. The supported hosting models are Blazor Server and Blazor WebAssembly.

You can specify your database connection string in the OnConfiguring overloaded method of the ProductDbContext class. However, in this implementation, you'll store the database connection settings in the AppSettings.json file and read it in the Program.cs file to establish a database connection.

Note that your custom data context class (the ProductDbContext class, in this example), must expose a public constructor that accepts an instance of type DbContextOptions<ApplicationDbContext> as an argument. This is needed to enable the runtime to pass the context configuration using a call to the AddDbContext() method to your custom DbContext class. The following code snippet illustrates how you can define a public constructor for your data context class.

```
public ProductDbContext
(DbContextOptions
<ProductDbContext> options,
IConfiguration configuration) :
base(options)
{
   _configuration = configuration;
}
```

Seed the Database

You might often want to work with data seeding when using Entity Framework Core (EF Core) to populate a blank database with an initial or minimal data set. Data seeding is a one-time process of loading data into a database. The EF Core framework provides an easy way to seed the data using the OnModelCreating() method of the DbContext class.

To generate fake data in your ASP.NET Core application, you can take advantage of the Bogus open-source library. It helps you to seed your database by taking advantage of randomly generated but realistic data. To use this library, you install the Bogus library from NuGet into your project. You can get it from here: https://www.nuget.org/packages/bogus.

The following code snippet illustrates how you can generate data using random data from the Bogus library.

```
private Product[]
GenerateProductData()
{
      var productFaker = new Faker
      <SCS.Product.Models.Product>()
      .RuleFor(x => x.Product_Id,
       f => Guid.NewGuid())
      .RuleFor(x =>
       x.Product_Name, f =>
       f.Commerce.ProductName())

       .RuleFor(x => x.Product_Description,
       f => f.Commerce.ProductDescription())

       .RuleFor(x => x.Product_Category,
       f => f.Commerce.ProductMaterial())

       .RuleFor(x => x.Product_Price,
       f => Math.Round(f.Random.Decimal
       (1000, 5000), 2));

     return productFaker.
     Generate(count: 5).ToArray();
}
```

Invoke the GenerateProductData method in the OnModelCreating method to populate the database with randomly generated data, as shown in the following piece of code:

```
protected override void
OnModelCreating
(ModelBuilder modelBuilder)
{
    modelBuilder.Entity
    <Models.Product> ().
    ToTable("Product");

    modelBuilder.Entity
    <Models.Product> ().
    HasKey(p => p.Product_Id);

    var products =
    GenerateProductData();

    modelBuilder.Entity
    <Models.Product> ().
    HasData(products);
}
```

The complete source code of the ProductDbContext class is given in **Listing 2**.

Register the ProductDb data context instance as a service with the services container of ASP.NET Core using the following piece of code in the Program.cs file:

```
builder.Services.AddDbContext
<ProductDbContext>(options =>
{
    options.UseSqlServer(
      builder.Configuration
   ["ConnectionStrings:DefaultConnection"]);
});
```

Listing 2: The ProductDbContext class

```
using Bogus;
using Microsoft.EntityFrameworkCore;
using SupplyChainManagementSystem.Shared.Models;

namespace SupplyChainManagementSystem.Server.Data
{

    public class ProductDbContext : DbContext
    {
        private readonly IConfiguration _configuration;

        public ProductDbContext(DbContextOptions
          <ProductDbContext>  options,
          IConfiguration configuration) : base(options)
        {
            _configuration = configuration;
        }


        protected override void
         OnConfiguring(DbContextOptionsBuilder
          optionsBuilder)
        {
            _ = optionsBuilder.UseSqlServer(_configuration.

              GetConnectionString("DefaultConnection")).
              EnableSensitiveDataLogging();
        }
        public DbSet<Product> Products { get; set; }
        protected override void OnModelCreating
        (ModelBuilder modelBuilder)
        {

            modelBuilder.Entity<Product>().ToTable("Product");
            modelBuilder.Entity<Product>().HasKey(p =>
              p.Product_Id);

            var products = GenerateProductData();
            modelBuilder.Entity<Product>().HasData(products);
        }

        private Product[] GenerateProductData()
        {
            var productFaker = new Faker<Product>()
                .RuleFor(x => x.Product_Id, f => Guid.NewGuid())
                .RuleFor(x => x.Product_Name, f =>
                  f.Commerce.ProductName())
                .RuleFor(x => x.Product_Description, f =>
                  f.Commerce.ProductDescription())
                .RuleFor(x => x.Product_Category, f =>
                  f.Commerce.ProductMaterial())
                .RuleFor(x => x.Product_Price, f =>
                  Math.Round(f.Random.Decimal(1000, 5000), 2));
            return productFaker.Generate(count: 5).ToArray();
        }
    }
}
```

If your application needs to perform multiple units of work, it's advisable to use a DbContext factory instead. To do this, register a factory by calling the AddDbContextFactory method in the Program.cs file of your project, as shown in the following code example:

```
builder.Services.AddDbContextFactory
<ProductDbContext>(options =>
{
    options.UseSqlServer
(builder.Configuration
["ConnectionStrings: DefaultConnection"]);
});
```

Create, Update, and Delete Products

Now that you know how to query data from the Product database table, let's understand how you can create a new product record, update an existing product record, and delete a product record from the database. To do this, you need to create commands to handle each of the Create, Update, and Delete operations.

Create the ProductRepository Class

A repository class is an implementation of the Repository design pattern and is one that manages data access. The application takes advantage of the repository instance to perform CRUD operations against the database. Now, create a new class named ProductRepository in a file having the same name with a .cs extension. Then write the following code in there:

```
public class ProductRepository :
IProductRepository
{
}
```

> In the model classes (both Product and Supplier), you've used record type. The reason is that a record type is an immutable, lightweight data type and much more efficient compared to a class type or even a struct type as far as performance and data integrity is concerned.

The ProductRepository class, illustrated in the code snippet below, implements the methods of the IProductRepository interface. Here is how the IProductRepository interface should look:

```
using SupplyChainManagementSystem.Shared.Models;
using System.Collections.Generic;

namespace SupplyChainManagementSystem.Server.Data
{
    public interface IProductRepository
    {
        public Task<List<Product>>
        GetAllProductsAsync();

        public Task<Product>
        GetProductByIdAsync(Guid id);

        public Task CreateAsync (Product product);
        public Task UpdateAsync (Product product);
        public Task DeleteAsync (Product product);
    }
}
```

Listing 3: The ProductRepository class

```
using SupplyChainManagementSystem.Shared.Models;

namespace SupplyChainManagementSystem.Server.Data
{
    public class ProductRepository
    {
        private readonly
        ProductDbContext _productDbContext;
        public ProductRepository(
            ProductDbContext productDbContext)
        {
            _productDbContext = productDbContext;
        }
        public async Task<List<Product>> GetProducts()
        {
            return await Task.FromResult(
                _productDbContext.Products.ToList());
        }

        public async Task<Product> GetProduct(Guid Id)
        {
            return await Task.FromResult
            (_productDbContext.Products.FirstOrDefault(
            x => x.Product_Id == Id));
        }
    }
}
```

Listing 4: The ProductController class

```
[Route("api/[controller]")]
[ApiController]
public class ProductController : Controller
{
    private IProductRepository _productRepository;
    public ProductController
    (IProductRepository productRepository)
    {
        _productRepository = productRepository;
    }

    [HttpGet("GetProducts")]
    public async Task<List<Product>> GetProducts()
    {
        return await _productRepository.GetProducts();
    }
}
```

The complete source code of the ProductRepository class is given in **Listing 3**.

Create the ProductController Class

Now, create a new controller named ProductController in the Controllers folder of the project. The following code snippet shows how you can take advantage of constructor injection to pass an instance of type IProductRepository interface using the constructor and then use it to retrieve all product records from the database.

```
public class ProductController : Controller
{
    private IProductRepository _productRepository;
    public ProductController(IProductRepository
    productRepository)
    {
        _productRepository = productRepository;
    }
}
```

The complete source of the ProductController class is given in **Listing 4**.

Create the Supplier Microservice

In this example, you'll build the Product microservice application or the Product API. The product microservice application is composed of the following files:

- **Supplier.cs:** The supplier model that contains domain-specific data and (optionally) business logic
- **ISupplierRepository.cs:** The ISupplierRepository interface that contains the declaration of the operations supported by the product repository
- **SupplierRepository.cs:** The supplier repository class that implements the members of the ISupplierRepository interface
- **SupplierDbContext.cs:** The supplier data context used to perform CRUD operations for the Supplier table in the database
- **appsettings.json:** The application's settings file where you can configure the database connection string, logging metadata, etc.
- **Program.cs:** Any ASP.NET Core application contains a file where the startup code required by the application resides. This file is named Program.cs where the services required by your application are configured. You can specify dependency injection (DI), configuration, middleware, and much more information in this file.

Create the Supplier Class

Create a new class named Order in a file having the same name with a .cs extension and write the following code in there:

```
public class Supplier
{
    public Guid Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Address { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
}
```

The other entity classes are not being shown here for brevity and also because this is a minimalistic implementation to illustrate a microservices-based application with just two modules: Product and Supplier.

Seed the Database

The following code snippet illustrates how you can generate random supplier data using random data from the Bogus library:

```
private Supplier[]
GenerateSupplierData()
{
   var supplierFaker =
   new Faker<Supplier>()

  .RuleFor(x => x.Supplier_Id,
  f => Guid.NewGuid())

  .RuleFor(x => x.Supplier_Name,
  f => f.Person.FullName)

  .RuleFor(x => x.Supplier_Address,
  f => f.Person.Address.ToString())

  RuleFor(x => x.Supplier_Phone,
  f => f.Person.Phone)

  .RuleFor(x => x.Supplier_Email,
  f => f.Person.Email);

  return supplierFaker.Generate
  (count: 5).ToArray();
}
```

Next, invoke the GenerateSupplierData method in the OnModelCreating method to populate the database with randomly generated supplier data, as shown in the following piece of code:

```
protected override void
OnModelCreating
(ModelBuilder modelBuilder)
{
     modelBuilder.Entity<Supplier>
     ().ToTable("Supplier");

    modelBuilder.Entity
    <Supplier>().HasKey(s =>
    s.Supplier_Id);

    var suppliers = GenerateSupplierData();
    modelBuilder.Entity<Supplier>
    ().HasData(suppliers);
}
```

The complete source code of the SupplierDbContext class is given in **Listing 5**.

Lastly, register the SupplierDb data context instance with the services container of ASP.NET Core using the following piece of code in the Program.cs file:

```
builder.Services.AddDbContext
<SupplierDbContext>
(options =>
{

   options.UseSqlServer(
   builder.Configuration
   ["ConnectionStrings:DefaultConnection"]);
});
```

Create the ISupplierRepository Interface

Create a new .cs file named ISupplierRepository in your project and replace the default generated code with the following code snippet:

```
public interface ISupplierRepository
{
    public Task<List<Supplier>> GetSuppliers();
    public Task<Supplier> GetSupplier(int Id);
}
```

Listing 5: The SupplierDbContext class

```
using Bogus;
using Microsoft.EntityFrameworkCore;
using SupplyChainManagementSystem.Shared.Models;

namespace SupplyChainManagementSystem.Server.Data
{
    public class SupplierDbContext : DbContext
    {
        private readonly IConfiguration _configuration;

        public SupplierDbContext(DbContextOptions
        <SupplierDbContext> options,
          IConfiguration configuration) : base(options)
        {
            _configuration = configuration;
        }

        protected override void
        OnConfiguring(DbContextOptionsBuilder
          optionsBuilder)
        {

            _ = optionsBuilder.UseSqlServer
              (_configuration.
              GetConnectionString("DefaultConnection")).
              EnableSensitiveDataLogging();
        }

        public DbSet<Supplier> Suppliers { get; set; }

        protected override void OnModelCreating
        (ModelBuilder modelBuilder)
        {

            modelBuilder.Entity<Supplier>().ToTable("Supplier");
            modelBuilder.Entity<Supplier>().HasKey(s =>
              s.Supplier_Id);

            var suppliers = GenerateSupplierData();
            modelBuilder.Entity<Supplier>().HasData(suppliers);
        }


        private Supplier[] GenerateSupplierData()
        {
            var supplierFaker = new Faker<Supplier>()
                .RuleFor(x => x.Supplier_Id,
                 f => Guid.NewGuid())
                .RuleFor(x => x.FirstName, f =>
                  f.Person.FirstName)
                 .RuleFor(x => x.LastName, f =>
                  f.Person.LastName)
                  .RuleFor(x => x.Address, f =>
                  f.Person.Address.ToString())
                .RuleFor(x => x.Phone, f =>
                  f.Person.Phone)
                .RuleFor(x => x.Email, f =>
                  f.Person.Email);

            return supplierFaker.Generate
            (count: 5).ToArray();
        }
    }
}
```

SPONSORED SIDEBAR:

AI Executive Briefing

Experience the game-changing impact of AI through CODE Consulting's Executive Briefing service. Uncover the immense potential and wide-ranging benefits of AI in every industry. Our briefing provides strategic guidance for seamless implementation, covering crucial aspects such as infrastructure, talent acquisition, and leadership.

Discover how to effectively integrate AI and propel your organization into future success.

Contact us today to schedule your executive briefing and embark on a journey of AI-powered growth. www.codemag.com/ai

Listing 6: The SupplierRepository class

```
using SupplyChainManagementSystem.Shared.Models;
namespace SupplyChainManagementSystem.Server.Data
{
    public class SupplierRepository : ISupplierRepository
    {
        private readonly
        SupplierDbContext _supplierDbContext;
        public SupplierRepository(
            SupplierDbContext supplierDbContext)
        {
            _supplierDbContext = supplierDbContext;
        }

        public async Task<List<Supplier>> GetSuppliers()
        {
            return await Task.FromResult(
                _supplierDbContext.Suppliers.ToList());
        }

        public async Task<Supplier> GetSupplier(Guid Id)
        {
            return await Task.FromResult
                (_supplierDbContext.Suppliers.
                 FirstOrDefault(
                   x => x.Supplier_Id == Id));
        }
    }
}
```

Listing 7: The SupplierController class

```
[Route("api/[controller]")]
[ApiController]
public class SupplierController : ControllerBase {
    private ISupplierRepository _supplierRepository;
    public SupplierController
    (ISupplierRepository supplierRepository)
    {
        _supplierRepository = supplierRepository;
    }

    [HttpGet("{id}")]
    public async Task<Supplier> GetSupplier(int id)
    {
        return await _supplierRepository.GetSupplier(id);
    }

    [HttpGet("GetSuppliers")]

    public async Task<List<Supplier>> GetSuppliers()

    {

        return await _supplierRepository.GetSuppliers();

    }

}
```

Create the SupplierRepository Class

Now create a new class named SupplierRepository in a file having the same name with a .cs extension. Write the following code in there:

```
public class SupplierRepository :
ISupplierRepository
{

}
```

The SupplierRepository class, illustrated in the code snippet below, implements the methods of the ISupplierRepository interface.

```
public async Task<List<Supplier>> GetSuppliers()
{
    return await Task.FromResult(suppliers);
}

public async Task<Order> GetSupplier(int Id)
{
  return await Task.FromResult(suppliers.
  FirstOrDefault(x => x.Id == Id));
}
```

The complete source code of the SupplierRepository class is given in **Listing 6**.

Create the SupplierController Class

Now, create a new controller named SupplierController in the Controllers folder of the project. The SupplierController exposes a couple of action methods to enable consuming one or more supplier records from a client application. Similar to the ProductController class, you'll leverage constructor injection to build an instance of the SupplierRepository class and then use it to retrieve supplier data from the database, as shown in the code snippet given below:

```
public class SupplierController :
Controller
{
    private ISupplierRepository
    _supplierRepository;

    public SupplierController
    (ISupplierRepository
    supplierRepository)
    {
        _supplierRepository =
       supplierRepository;
    }
}
```

The complete source of the SupplierController class is given in **Listing 7**.

Register the Service Instances with IServiceCollection

The following code snippet illustrates how you can register the IRequestHandler instances added as a transient service to the IServiceCollection:

```
builder.Services.AddScoped
<IProductRepository,
ProductRepository>();
```

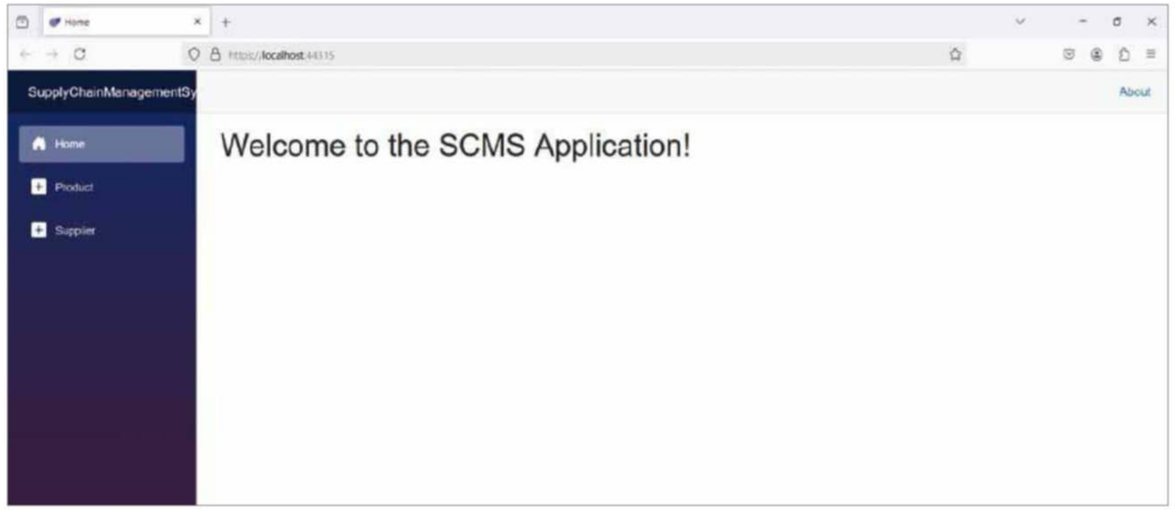
Register the IProductRepository instance with the service collection, as shown below:

```
builder.Services.AddScoped
<IProductRepository, ProductRepository>();
```

The following code snippet illustrates how an instance of type ISupplierRepository is added as a scoped service to the IServiceCollection:

Listing 8: The Program.cs file (Server/API project)

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using SupplyChainManagementSystem.Server.Data;
using SupplyChainManagementSystem.Server.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddDbContext<ApplicationDbContext>(
    options => options.UseSqlServer(
        builder.Configuration.GetConnectionString
("DefaultConnection")));

builder.Services.AddIdentity<ApplicationUser,
IdentityRole>().AddEntityFrameworkStores
<ApplicationDbContext>();

builder.Services.AddScoped<IProductRepository,
ProductRepository>();

builder.Services.AddScoped<ISupplierRepository,
SupplierRepository>();

var app = builder.Build();

// Configure the HTTP request pipeline.

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.Run();
```



Welcome to the SCMS Application!

Figure 13: The SupplyChainManagementSystem application in execution

```
builder.Services.AddScoped
<ISupplierRepository,
SupplierRepository>();
```

The complete source code of the Program.cs file is given in **Listing 8**.

> You can implement authentication in your Blazor application using the built-in authentication provided by Microsoft. If you already have a Blazor application, you can't follow this approach to implement authentication in your Blazor applications. It's preferable to implement an out-of-the-box authentication to overcome the limitations of this approach.

Execute the SupplierChain-ManagementSystem Application

Finally, run the application by pressing the F5 key on your keyboard. When the application executes, you'll observe the Home page displayed, as shown in **Figure 13**.

Where Do We Go from Here?

Blazor is a feature-rich contemporary framework and cutting-edge technology that avoids dealing with the nitty-gritty of developing interactive web apps. It supports both client-side and server-side hosting models, providing flexibility, enhanced performance, and the ability to build full-stack applications in .NET and C#.

Joydip Kanjilal

Boost Your iOS App Development with ChatGPT AI-Assistance

The genie's out of the bottle and there's no turning back! If you haven't tried AI-assisted programming yet, now is a great time to dive in. Tools such as ChatGPT have come a long way from their inception. You'll be amazed at how much more productive you'll be and wonder how you ever managed without them. The genie's out of the bottle and there's no turning back!

Kevin J. McNeish
kevin@oakleafdigitalsoftware.com
www.oakleafdigitalsoftware.com/blog
@twitter.com/kjmcneish

Kevin McNeish is author of "Coding iOS Apps with ChatGPT" and numerous other books and articles. A seven-time Microsoft MVP awardee, Kevin is an accomplished international conference speaker and President of Oak Leaf Digital, a software company specializing in innovative solutions. Kevin also teaches in-person seminars, empowering developers to maximize the potential of ChatGPT in their work.

If you haven't tried AI-assisted programming yet, now is a great time to dive in. Tools such as ChatGPT have come a long way from their inception. You'll be amazed at how much more productive you'll be and wonder how you ever managed without them.

In this article, I explore practical applications of ChatGPT for iOS app development, debunk common myths, and showcase its potential and pitfalls as a coding partner. Whether you're new to ChatGPT or already familiar with it, you'll find valuable information about recently released features, including Projects and Canvas Mode. This article specifically highlights the capabilities of ChatGPT Plus and GPT-4 Turbo, unlocking new levels of efficiency and creativity.

ChatGPT: Fear and Loathing

I've encountered some negative reactions when discussing AI-assisted programming with some developers. This resistance is a common response to disruptive technologies. One argument I often hear is "A developer worth their salt doesn't need ChatGPT." I believe this sentiment stems in part from the negative publicity ChatGPT received early on, including stories of students using it to write essays, complete assignments, and pass exams, passing off the AI-generated work as their own. As a result, some developers equate using ChatGPT with cheating or, worse, as an admission of inadequacy and incompetence.

In response, I make the point that ChatGPT is just another tool in a developer's arsenal. It helps you perform your usual tasks faster and more accurately. Not using ChatGPT is like refusing to upgrade to a newer compiler that's better at finding coding errors—it's a missed opportunity to enhance your productivity and code quality. Just as an improved compiler can catch subtle bugs, suggest optimizations, or provide clearer error messages, ChatGPT serves as a coding assistant that improves your efficiency. It can help debug tricky issues, generate boiler plate code, or even clarify complex programming concepts, giving you more time to focus on solving challenging problems.

Then there's the fear that AI will make coders obsolete. I talked to one aspiring programmer who decided not to pursue a career in coding believing that tools like ChatGPT would handle all coding tasks, leaving little need for human input. I think this speaks to a lack of awareness of AI's limitations. AI is far from replacing the creative, human-centric aspects of development. Programming often involves deeply understanding business needs, solving ambiguous problems, and crafting creative and efficient solutions—areas where human developers excel.

ChatGPT enhances efficiency and creativity, but doesn't replace critical thinking or domain expertise.

> ChatGPT enhances efficiency and creativity, but doesn't replace critical thinking or domain expertise.

Ultimately, ChatGPT doesn't "think" or innovate; it assists with repetitive or time-consuming tasks, leaving humans to focus on innovation and strategy. Understanding what AI encompasses helps contextualize its role. Artificial intelligence refers to systems or machines that **simulate** human intelligence to perform tasks. AI can be categorized into:

- **Narrow AI (Weak AI):** Focused on specific tasks, such as language processing, or image recognition. ChatGPT falls into this category.
- **General AI (Strong AI):** A hypothetical future system capable of performing any intellectual task a human can do, including reasoning and self-awareness.

ChatGPT is **Narrow AI,** specializing in generating and understanding natural language rather than possessing human-like intelligence.

Throughout this article and in those to come, I'll highlight the limitations and strengths of using AI as a tool. Ultimately, ChatGPT isn't coming for your job, but people who know how to leverage it are.

> ChatGPT isn't coming for your job, but developers who know how to leverage it are.

ChatGPT Free vs. Paid Version

Although there's a free version of ChatGPT available, I highly recommend subscribing to ChatGPT Plus for $20 per month. The free version (GPT-3.5) has decent capabilities, but the tools you really need are in GPT-4 Turbo with ChatGPT Plus.

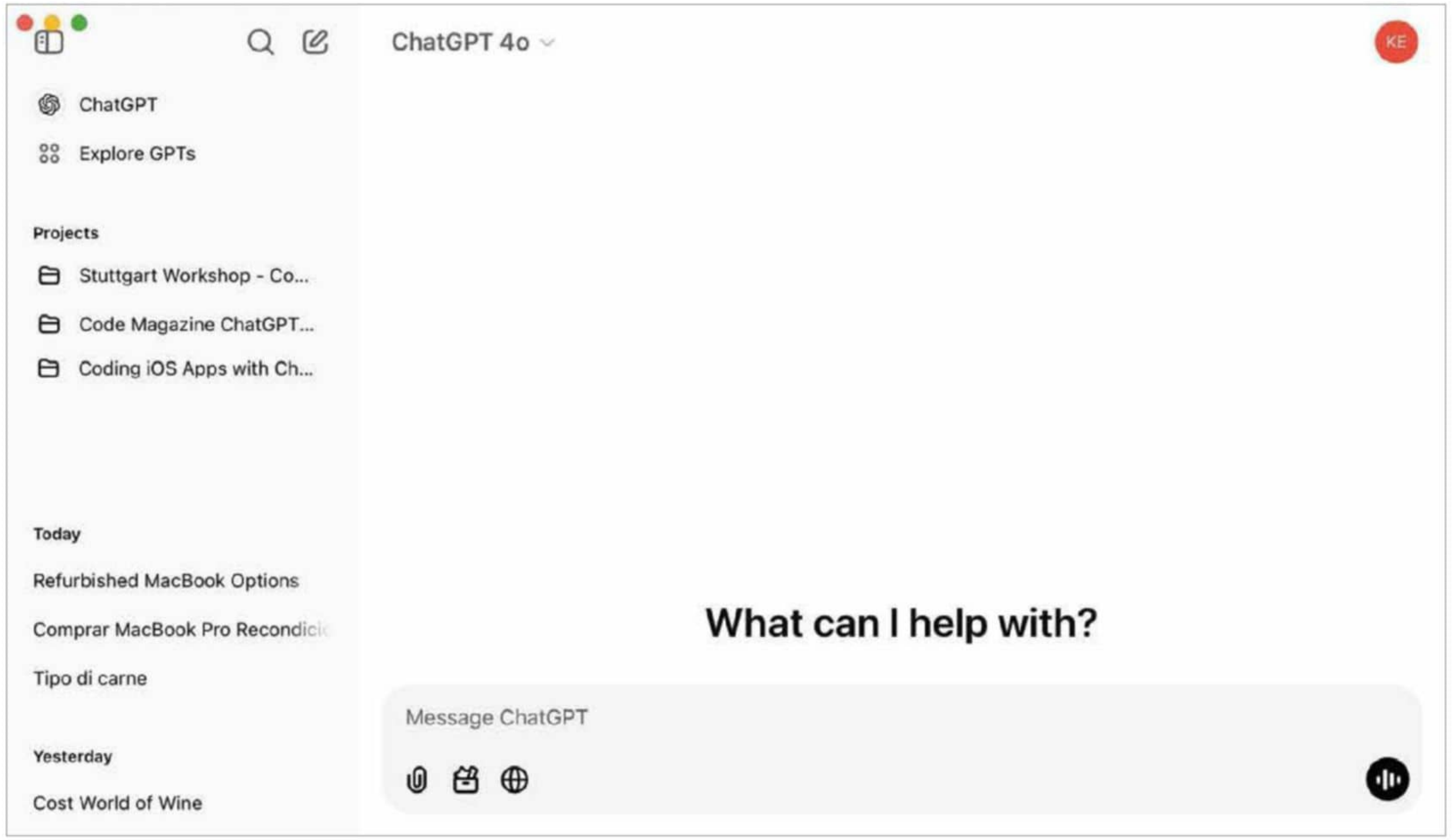


Figure 1: ChatGPT lists the projects you create in its left-hand side bar.

Here are the extras you get with ChatGPT Plus that you don't get with the free version, which I outline in more detail in the next section:

- **Projects:** Organizes and manages work by creating persistent, structured spaces for related tasks, documents, and code. It's ideal for long-term project like app development.
- **Coding Assistance:** It possesses far more comprehensive understanding and debugging of complex code, refactoring large codebases, and designing intricate software patterns.
- **Larger Context Window:** The free version of ChatGPT can only handle shorter snippets of code (approximately 4,096 tokens or pieces of text). In contrast, ChatGPT-4 Turbo processes 128,000 tokens, handling longer code files and project discussions.
- **Image Inputs:** Analyzes and responds to questions about uploaded images.
- **Document Analysis:** Reviews and extracts information from uploaded documents.
- **Advanced Coding Assistance:** Has enhanced reasoning capabilities for complex coding tasks and debugging.
- **DALL-E Image Generation:** Creates and edits images directly in the chat, including editing specific parts of an image.
- **Canvas Mode:** Provides a dedicated workspace for editing long-form content or working on code collaboratively.

Given the obvious advantages over the free version, this article focuses on ChatGPT-4 Turbo. Let's take a closer look at some of these features and how they can enhance your app development.

Working with the Projects Feature

Projects, introduced on December 13, 2024 during **OpenAI's 12 Days of OpenAI** event, provide a structured workspace for managing long-term or multi-step tasks. Projects lets you keep related work organized, including files and datasets, as well as group multiple conversations-related topics. ChatGPT Plus users can create an unlimited number of projects.

In **Figure 1**, you can see a list of projects in the left panel of the ChatGPT Desktop window.

I've created projects for:

- My book, ***Coding iOS Apps with ChatGPT***
- A training seminar I'm giving in Germany
- This series of CODE Magazine articles

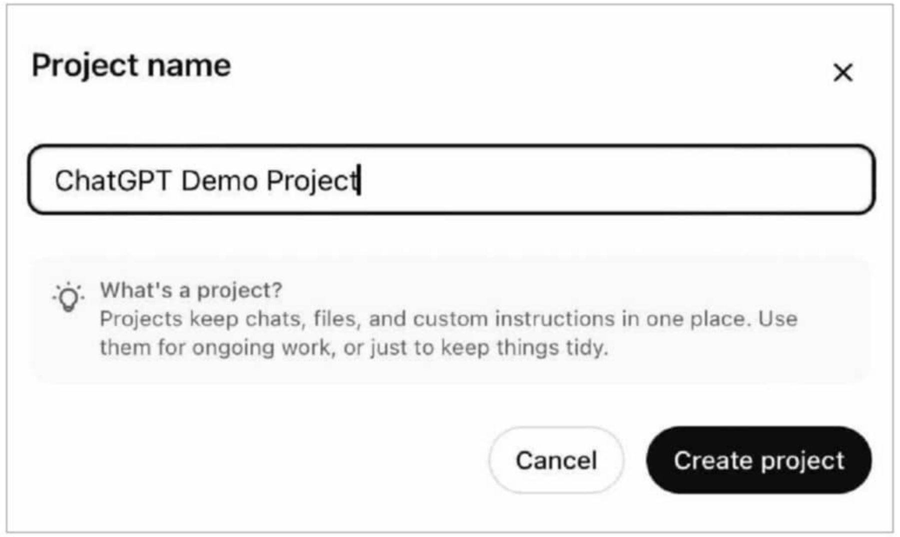
These folders allow me to keep the different projects I'm working on organized. When I switch tasks during my workday, I just click the project I want to work on and begin interacting with ChatGPT.

Let's create a new project to see how this works.

1. Hover your mouse over **Projects** in the sidebar and click the plus (+) button that appears. This displays a New Project dialog, as shown in **Figure 2**.

2. Enter the name of the project (for example, **ChatGPT Demo Project**).
3. Click the **Create project** button.

This displays your new project in the Project Panel on the right (**Figure 3**). At the top-left of the Project Panel is a folder. If you hover your mouse over this folder, a pencil icon appears that, when clicked, allows you to set the color of the project folder in the side panel. You can also hover over the project name, click the pencil icon that appears and change the name of the project.



Project name

ChatGPT Demo Project

What's a project?
Projects keep chats, files, and custom instructions in one place. Use them for ongoing work, or just to keep things tidy.

Figure 2: ChatGPT Plus-tier users can create an unlimited number of projects.

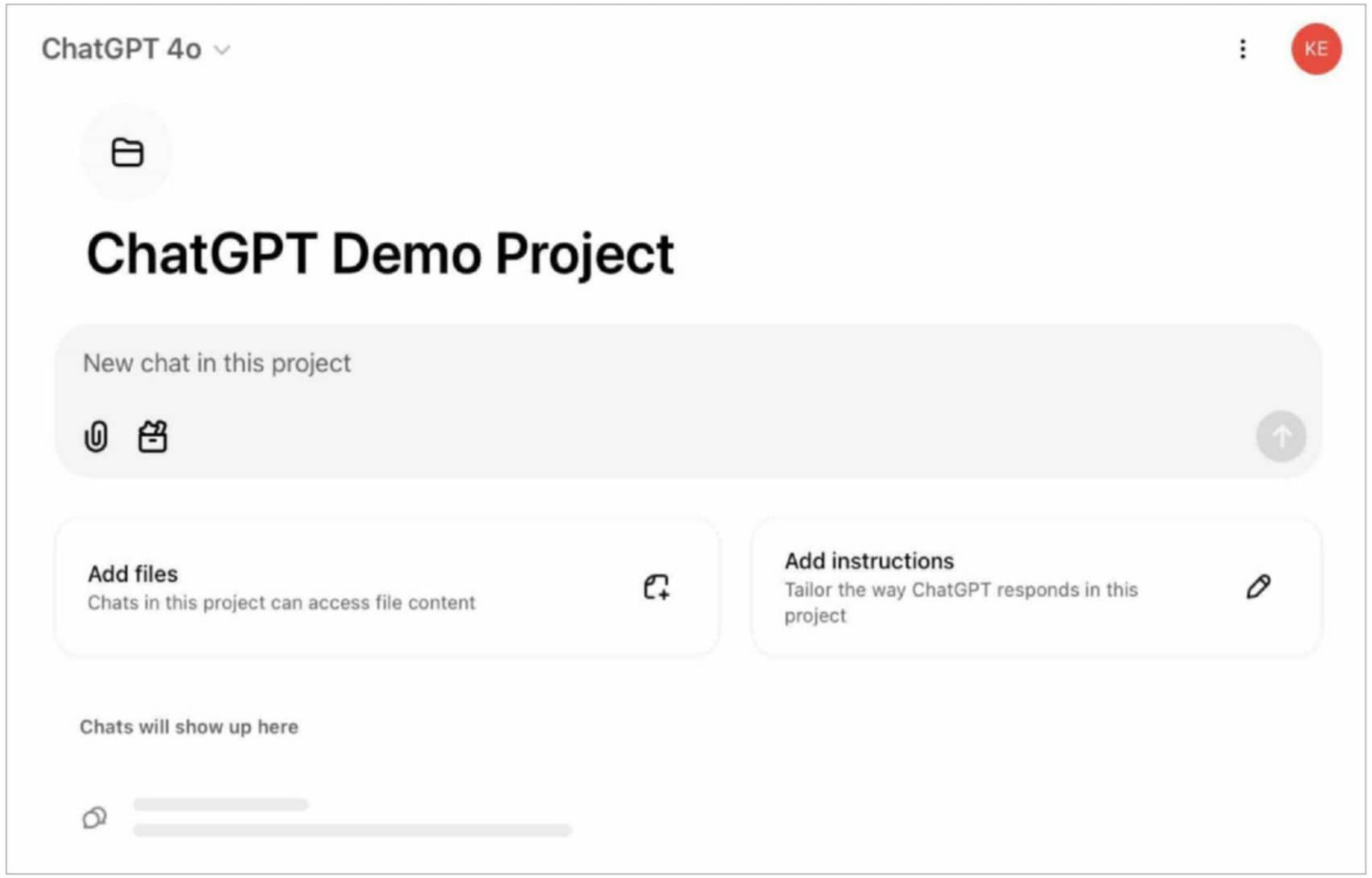
Adding Project Instructions

You can tailor ChatGPT's responses within a project by adding instructions. For example:

- Follow Swift API Design Guidelines, using camelCase for variables and PascalCase for classes and structs.
- The app must support iOS 17 and later.
- Use SwiftUI for building the user interface, with reusable components.
- Use Alamofire for networking.
- Use ObservableObject for state management.
- Ensure offline caching is implemented using CoreData.

After specifying instructions, ChatGPT actively uses them as a reference when answering project-specific questions. Let's try out this feature.

1. In the Project panel, click the **Add Instructions** button.
2. In the **Instructions** dialog (**Figure 4**), enter **When answering questions about the JSON files in this project, display responses in YAML format unless specified otherwise**.



ChatGPT 4o

ChatGPT Demo Project

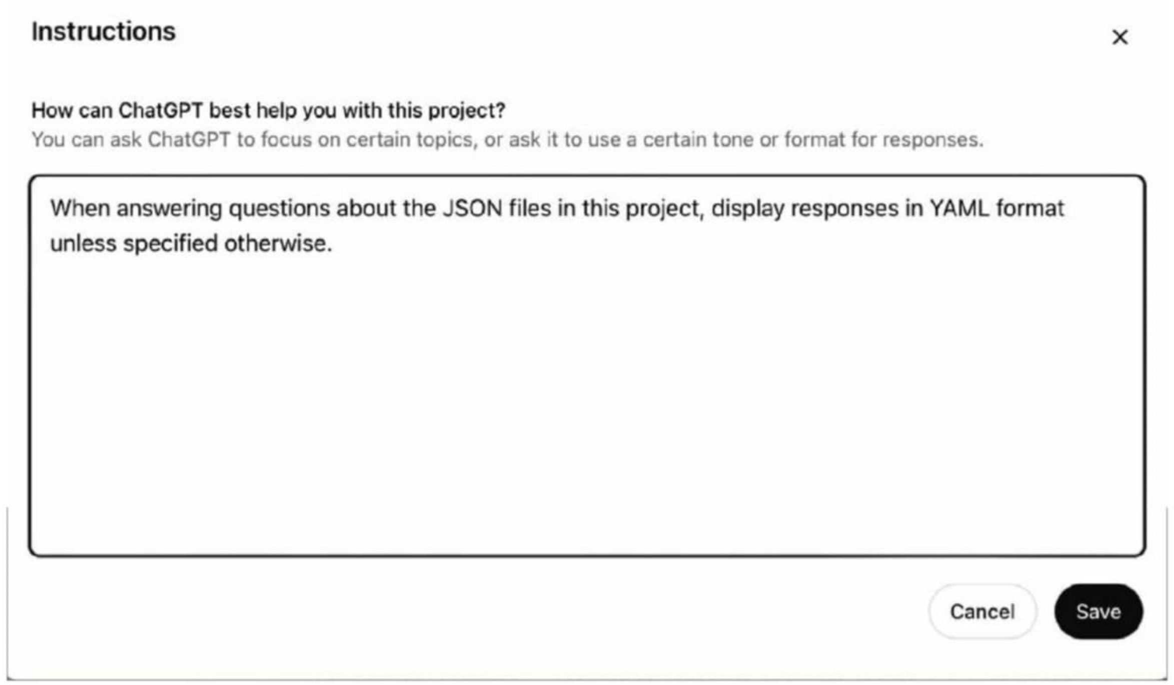
New chat in this project

Add files
Chats in this project can access file content

Add instructions
Tailor the way ChatGPT responds in this project

Chats will show up here

Figure 3: Creating or selecting a project displays the Project Panel on the right.



Instructions

How can ChatGPT best help you with this project?
You can ask ChatGPT to focus on certain topics, or ask it to use a certain tone or format for responses.

When answering questions about the JSON files in this project, display responses in YAML format unless specified otherwise.

Figure 4: Add instructions to tailor the way ChatGPT responds to questions in your project.

3. Click **Save** to add this instruction to the project. This displays the first several words of the instruction in the **Instructions** UI control.

In the next section, you'll add a JSON file for ChatGPT to analyze and process so you can see this instruction at work.

> You can tailor ChatGPT's responses within a project by adding instructions.

Adding Project Documents and Files

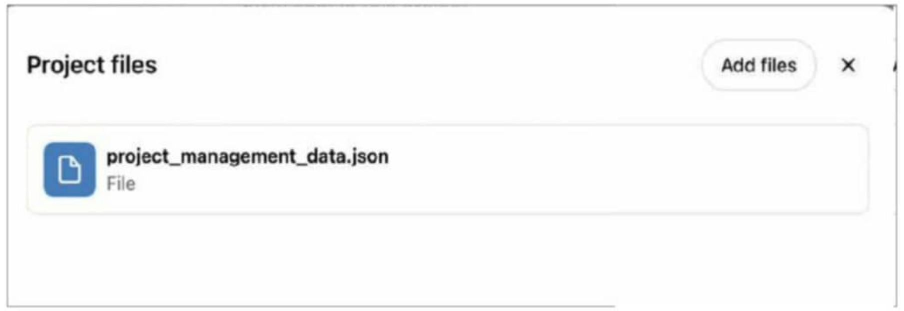
In October 2023, OpenAI expanded ChatGPT's ability to interpret and respond to visual inputs such as diagrams, screenshots, and handwritten notes—features I use every day. For example, I can take screenshots of several pages of mobile app exceptions logged in Sentry.io and ask ChatGPT for insights, like "What percentage of errors are being caused by nullable exceptions?" ChatGPT's answers help me get the big picture of where I should concentrate my efforts. Similarly, I can upload a screenshot of an Xcode Instruments Performance Report to identify high memory usage or excessive CPU consumption and get optimization recommendations.

ChatGPT excels at processing uploaded documents, such as PDFs and Word documents, offering capabilities for extracting, summarizing, and analyzing data. For example, although ChatGPT didn't write my latest book, I uploaded the draft Word document and asked it to check for spelling and grammatical errors. Its multi-lingual abilities are also impressive; I recently, received PDF documents in German from two hotels for a seminar I'm holding in Germany. I uploaded both documents and asked ChatGPT to give me an overview of each contract to help me decide on the venue. It did a great job of translating the documents into English and giving me a list of pros and cons for each hotel.

If you want to use this feature for your projects, the **Add files** option in ChatGPT's Projects panel is the best way to upload documents or datasets. Let's kick the tires on this feature. In the download files for this article (on www.CODEMag.com), there's a **project_data.json** file that contains a fictional dataset about projects, tasks, and team members.

Here are the steps to upload a file:

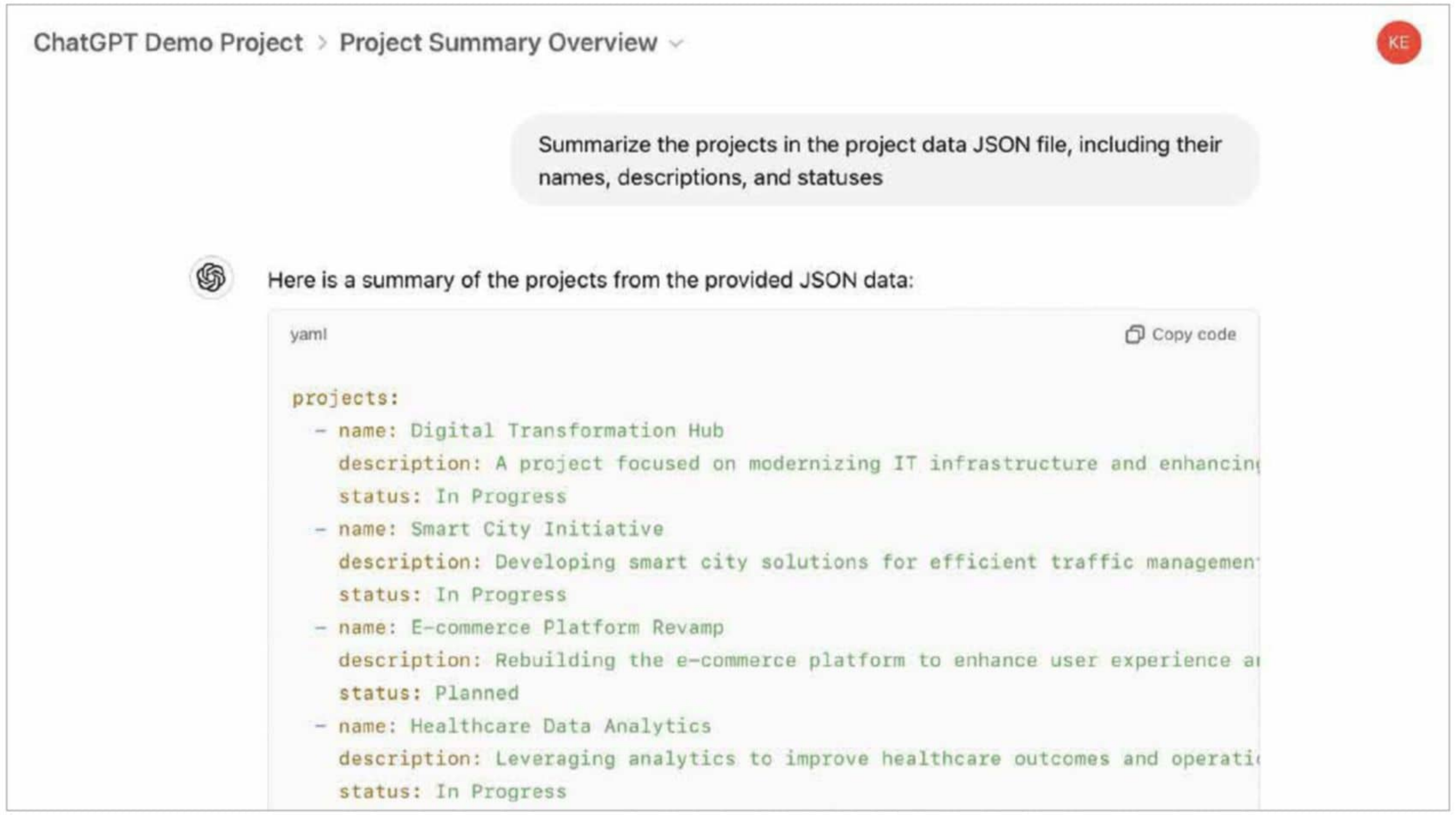
1. In the Projects panel, click the **Add files** button to open the **Project files** dialog (**Figure 5**).
2. Click the **Add files** button.
3. Navigate to the folder that contains the **project_data.json** file, select it, and then click **Open**.
4. An upload progress indicator appears. Once the upload is complete, an icon indicating the file type is displayed.
5. Close the dialog by clicking the **X** in the top-right corner.



Project files

project_management_data.json
File

Figure 5: You can upload documents and files associated with a project for ChatGPT to analyze.



Summarize the projects in the project data JSON file, including their names, descriptions, and statuses

Here is a summary of the projects from the provided JSON data:

```yaml
projects:
  - name: Digital Transformation Hub
    description: A project focused on modernizing IT infrastructure and enhancin
    status: In Progress
  - name: Smart City Initiative
    description: Developing smart city solutions for efficient traffic managemen
    status: In Progress
  - name: E-commerce Platform Revamp
    description: Rebuilding the e-commerce platform to enhance user experience a
    status: Planned
  - name: Healthcare Data Analytics
    description: Leveraging analytics to improve healthcare outcomes and operati
    status: In Progress
```

Figure 6: Analyze file contents by asking ChatGPT specific questions.

After uploading, the UI displays: **Project files – 1 file**.

With the file uploaded, you can query its contents. Try entering the following command in the Chat Input field: **Summarize the projects in the project data JSON file, including their names, descriptions, and statuses.**

Click the **Send** button (up arrow) in the bottom-right corner of the Chat Input Field to send the request.

ChatGPT responds by summarizing the projects in YAML format, as specified in the project instructions (**Figure 6**).

You can also try questions like:

- Show me all team members and their roles.
- Show me all tasks assigned to John Doe.
- What projects haven't started yet?
- Calculate the total duration of each project.

These features make ChatGPT a power tool for analyzing project data and extracting actionable insights from uploaded files.

Canvas Mode

Canvas Mode is one of ChatGPT's most powerful features, providing a dedicated workspace for editing long-form content or collaborating on code. This mode is especially useful for managing complex projects, providing developers with tools to organize and visualize their work.

Here are some of the key features of Canvas Mode:

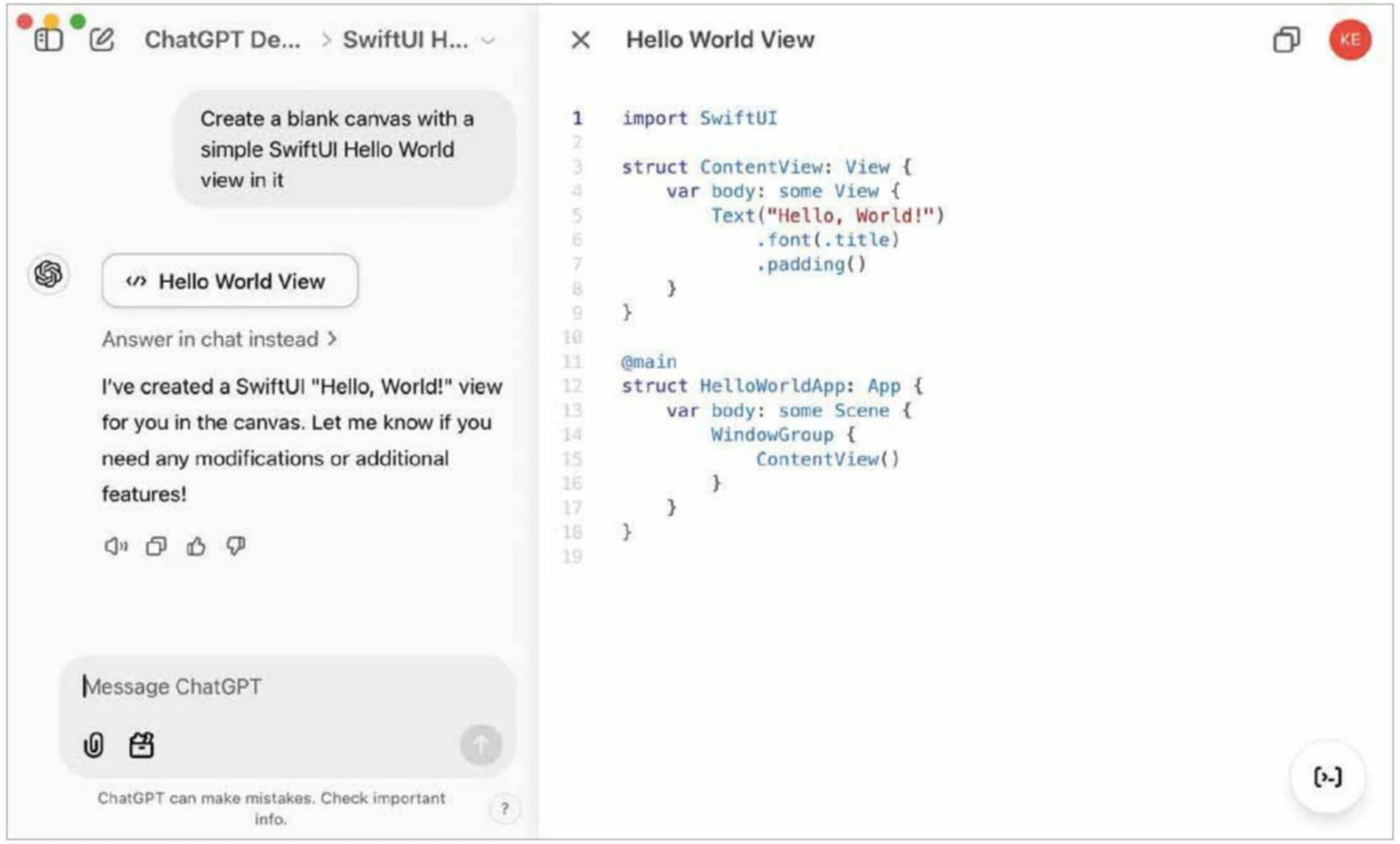
- Focused Editing: Canvas Mode allows you to view and edit your code or content in a distraction-free environment, which is ideal for managing extensive codebases or lengthy documentation.
- Collaboration: If you have an enterprise version of ChatGPT, you can collaborate with team members by sharing the canvas workspace. Changes appear in real-time, making it easier to co-develop or review.
- Version Tracking: Canvas Mode saves changes as iterations, making it easy to revert to previous versions. This feature is crucial for maintaining code integrity during iterative development.
- Enhanced Visualization: With syntax highlighting, formatting options, and structured file organization, Canvas Mode makes it easy to navigate large projects.

To create a new canvas, type the following in the ChatGPT chat window: **Create a blank canvas with a simple SwiftUI Hello World view in it.**

When you do this, the ChatGPT sidebar disappears, the Chat window takes its place, and the canvas appears on the right side of the screen, as shown in **Figure 7**.

Now you can manually edit the code in the canvas or ask ChatGPT to make modifications. For example, you could type: **Add a navigation bar to this view.**

ChatGPT animates the code update, showing changes line by line as it works. The newly added **NavigationView** appears in the canvas, as shown in **Figure 8**.

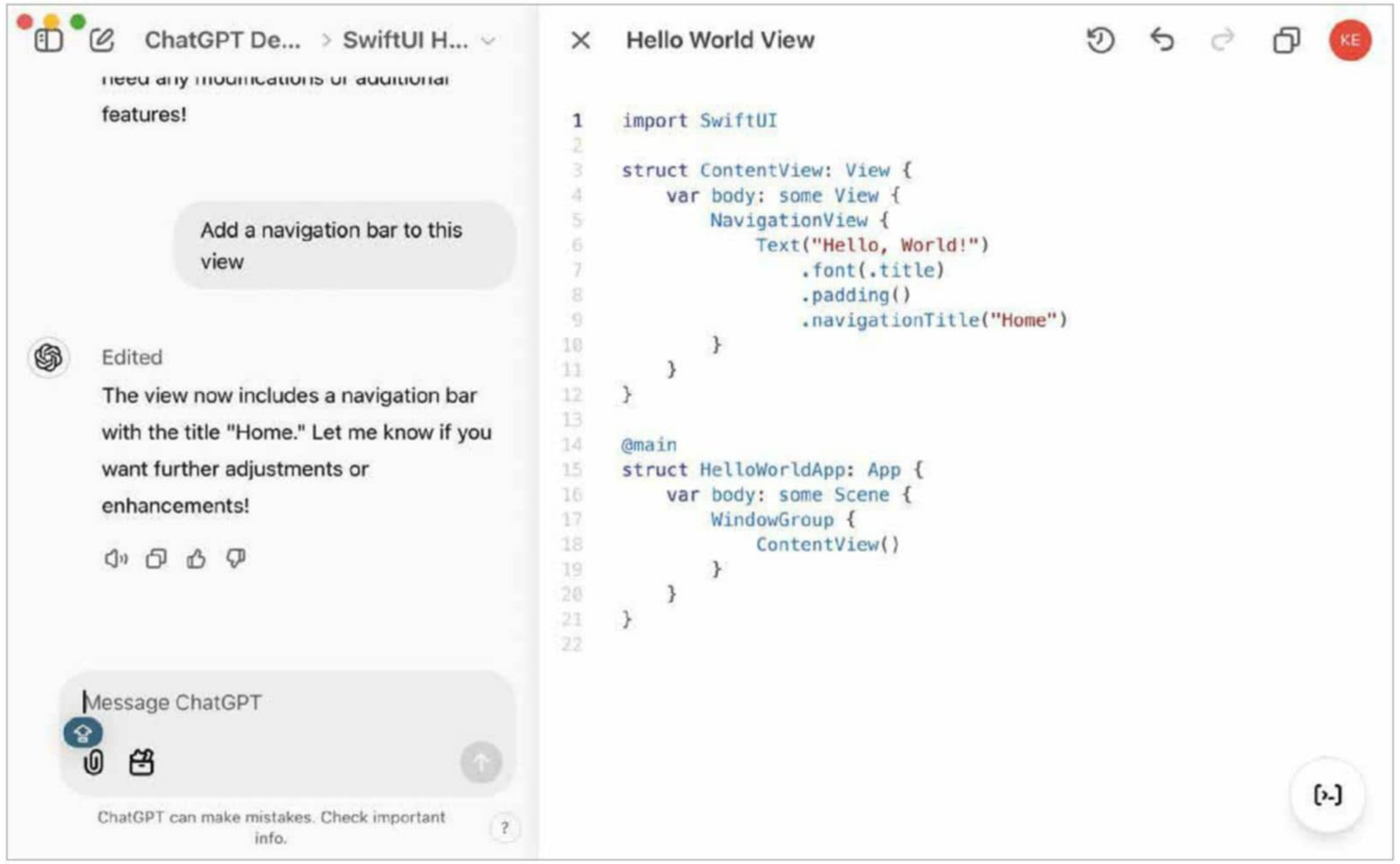

ChatGPT De... > SwiftUI H...
Hello World View
KE
Hello World View
Answer in chat instead >
Message ChatGPT
ChatGPT can make mistakes. Check important info.
?

Create a blank canvas with a simple SwiftUI Hello World view in it

I've created a SwiftUI "Hello, World!" view for you in the canvas. Let me know if you need any modifications or additional features!

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Hello, World!")
            .font(.title)
            .padding()
    }
}

@main
struct HelloWorldApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

Figure 7: The new ChatGPT canvas provides a dedicated workspace for editing long-form content.


ChatGPT De... > SwiftUI H...
Hello World View
KE
Edited
Message ChatGPT
ChatGPT can make mistakes. Check important info.
?

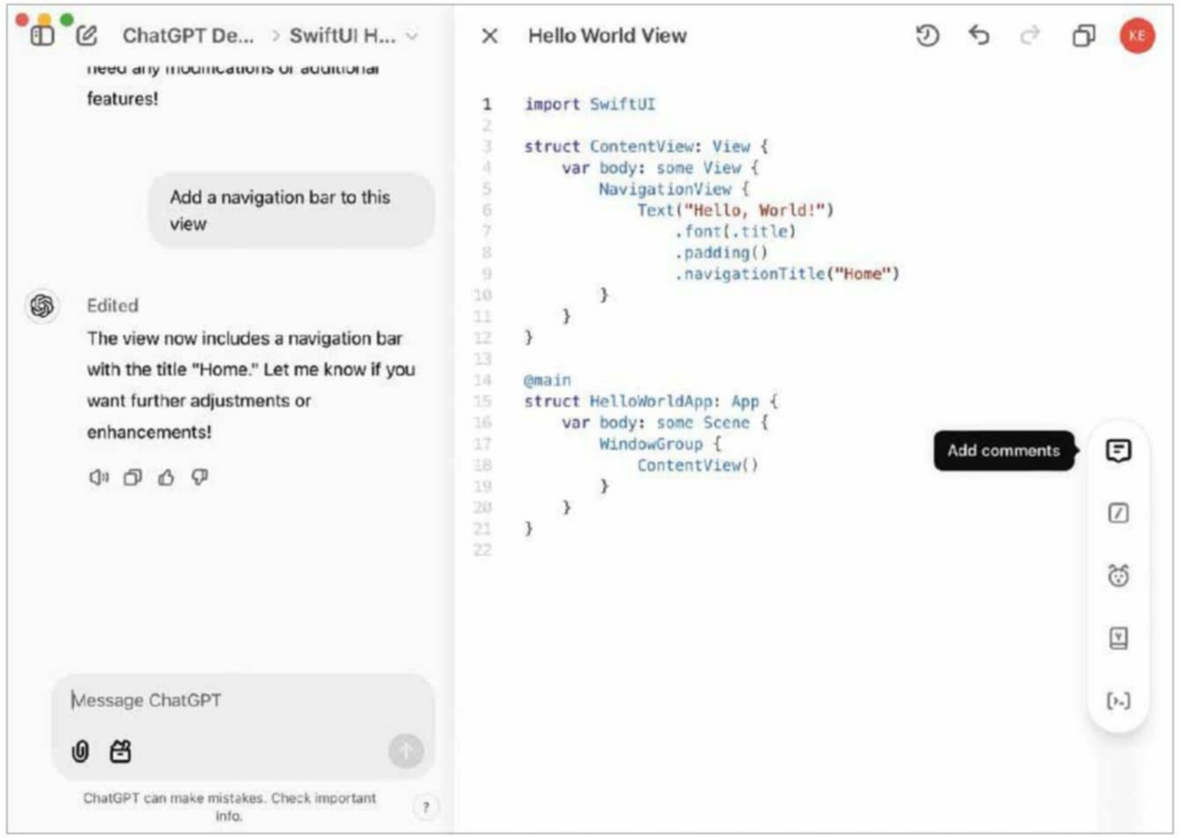
need any modifications or additional features!

Add a navigation bar to this view

The view now includes a navigation bar with the title "Home." Let me know if you want further adjustments or enhancements!

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        NavigationView {
            Text("Hello, World!")
                .font(.title)
                .padding()
                .navigationTitle("Home")
        }
    }
}

@main
struct HelloWorldApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

Figure 8: ChatGPT animates code updates in a canvas.



need any modifications or additional features!

Add a navigation bar to this view

Edited

The view now includes a navigation bar with the title "Home." Let me know if you want further adjustments or enhancements!

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        NavigationView {
            Text("Hello, World!")
                .font(.title)
                .padding()
                .navigationTitle("Home")
        }
    }
}

@main
struct HelloWorldApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

Figure 9: The canvas action menu gives easy access to key features.

At the bottom right corner of the canvas is an action menu. Hover over it to access several options (**Figure 9**).

- **Add Comments:** Automatically inserts comments into your code in the canvas
- **Add Logs:** Adds **print()** statements to your code to track key events in your app
- **Fix Bugs:** Analyzes the code file line by line to detect and fix bugs
- **Port to a Language:** Opens a popup to select a target language: PHP, C++, Python, JavaScript, TypeScript, Java. It then creates a new canvas with the converted code.
- **Code Review:** Reviews the code line by line, adding comments with suggestions for improvement

For instance, selecting **Add Comments** and **Add Logs** results in automatically annotated code with well-placed log statements, as shown in **Figure 10**. Although the comments exceed what you normally write, the logging statements are well placed.

The **Code Review** is an interesting option—especially when ChatGPT reviews the code it generated itself. **Figure 11** shows the result of this code review. As you can see, ChatGPT highlights specific lines of code and offers improvement suggestions in a balloon on the right. I've worked extensively with ChatGPT over the last few years in both Swift and C# code, and it's surprisingly good at finding issues with your code.

At the top-right corner of the canvas, version control buttons allow you to go navigate between iterations of your code or copy the current version to your clipboard. These tools ensure that you can easily track your work.

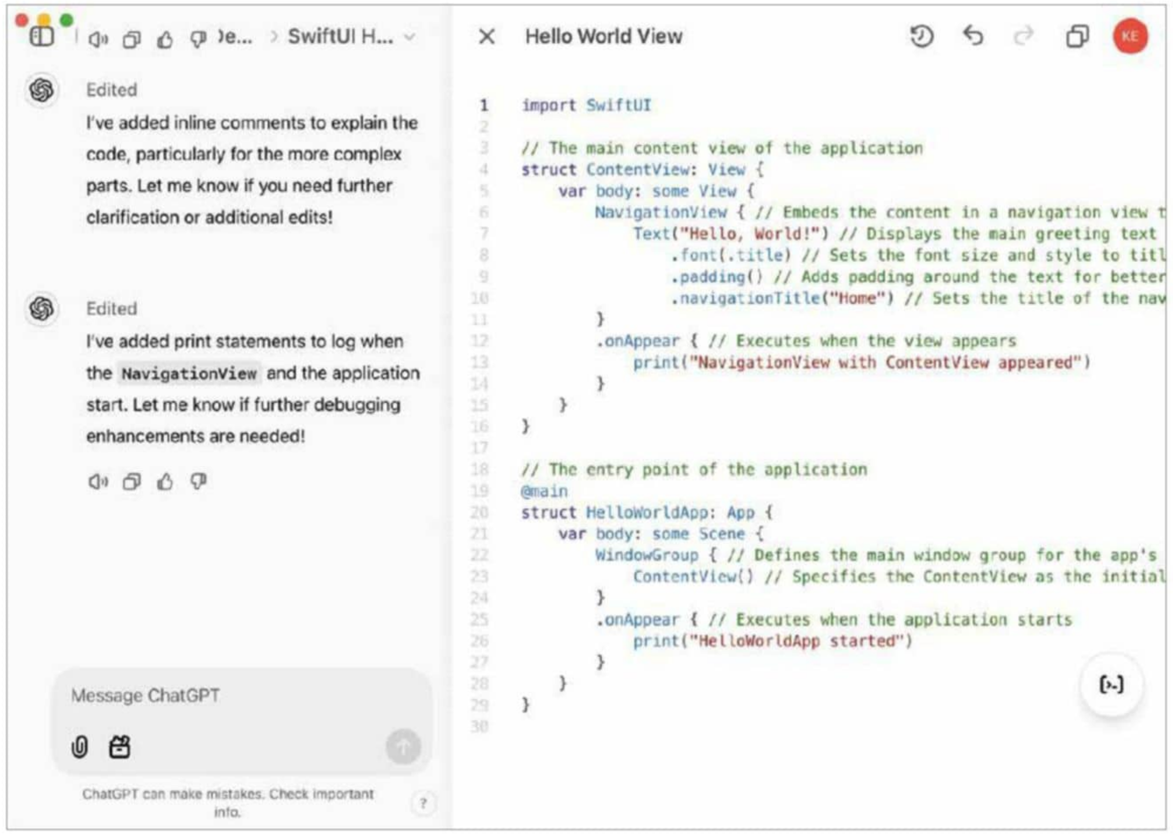
All in all, Canvas Mode allows ChatGPT to not just be a coding assistant, but a part of the development process. Whether you're editing, debugging, or porting code, the features in Canvas Model make it a great tool for developers.

Holding ChatGPT Accountable

For all of its virtues, there are times when ChatGPT can be very frustrating. It's perfectly capable of generating code that contains critical errors that prevent compilation. Fortunately, in most cases, it provides a corrected solution. However, there are times when resolving issues requires more persistence and context.

I was working on some low-level C# code today, and ChatGPT was stuck in a loop of producing code that wouldn't compile. To get around this, I located the relevant .NET class-level code file, copied it, and pasted it into the chat. With that additional context, ChatGPT finally produced code that worked without a hitch. I was curious, so I asked ChatGPT why it didn't know about the existence of the method I needed. It replied:

> That's a great question! The reason I initially missed the method is that I don't have direct access to every library's source code or detailed documentation unless provided during our conversation. My suggestions are based on my training, publicly available knowledge, and the context you provide.



Edited

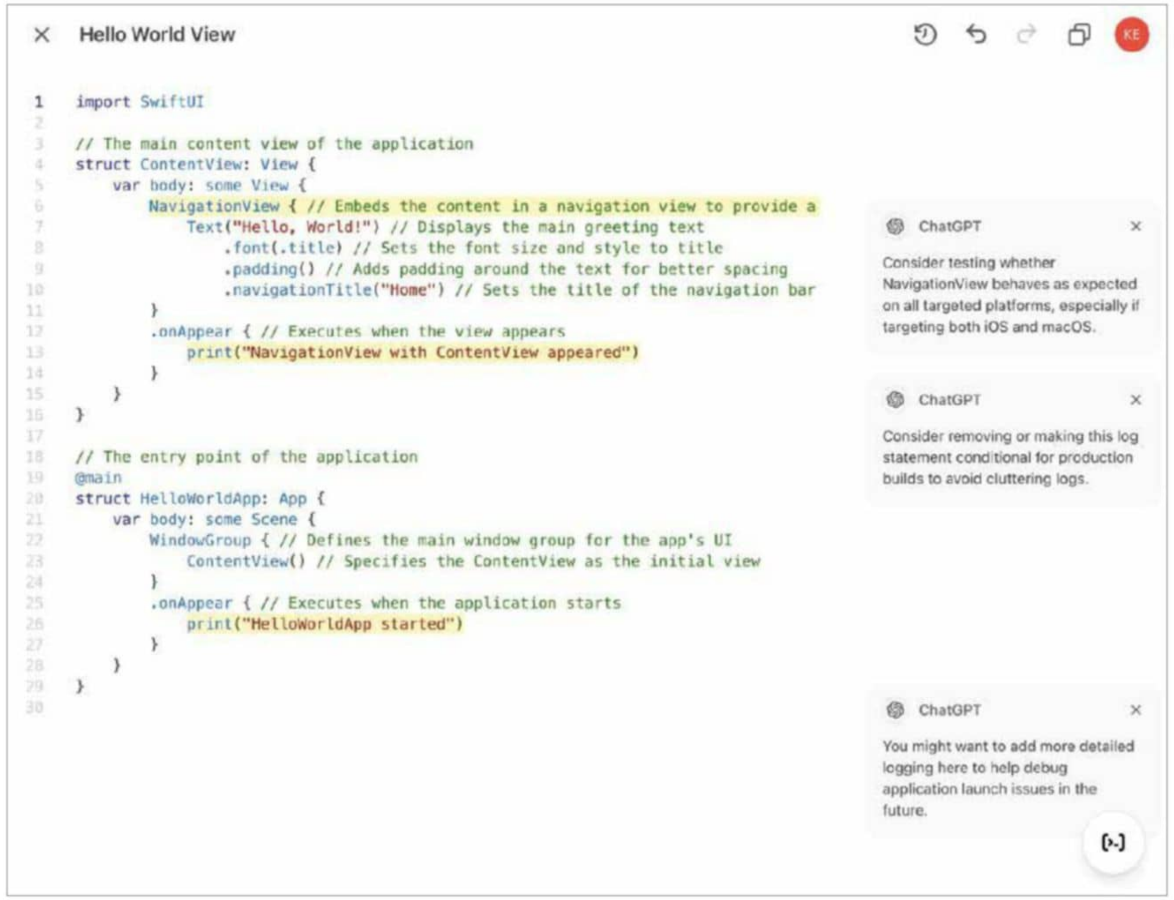
I've added inline comments to explain the code, particularly for the more complex parts. Let me know if you need further clarification or additional edits!

Edited

I've added print statements to log when the `NavigationView` and the application start. Let me know if further debugging enhancements are needed!

```
import SwiftUI

// The main content view of the application
struct ContentView: View {
    var body: some View {
        NavigationView { // Embeds the content in a navigation view t
            Text("Hello, World!") // Displays the main greeting text
                .font(.title) // Sets the font size and style to titl
                .padding() // Adds padding around the text for better
                .navigationTitle("Home") // Sets the title of the nav
        }
        .onAppear { // Executes when the view appears
            print("NavigationView with ContentView appeared")
        }
    }
}

// The entry point of the application
@main
struct HelloWorldApp: App {
    var body: some Scene {
        WindowGroup { // Defines the main window group for the app's
            ContentView() // Specifies the ContentView as the initial
        }
        .onAppear { // Executes when the application starts
            print("HelloWorldApp started")
        }
    }
}
```

Figure 10: ChatGPT can automatically add comments and log statements to your code.



```
import SwiftUI

// The main content view of the application
struct ContentView: View {
    var body: some View {
        NavigationView { // Embeds the content in a navigation view to provide a
            Text("Hello, World!") // Displays the main greeting text
                .font(.title) // Sets the font size and style to title
                .padding() // Adds padding around the text for better spacing
                .navigationTitle("Home") // Sets the title of the navigation bar
        }
        .onAppear { // Executes when the view appears
            print("NavigationView with ContentView appeared")
        }
    }
}

// The entry point of the application
@main
struct HelloWorldApp: App {
    var body: some Scene {
        WindowGroup { // Defines the main window group for the app's UI
            ContentView() // Specifies the ContentView as the initial view
        }
        .onAppear { // Executes when the application starts
            print("HelloWorldApp started")
        }
    }
}
```

Consider testing whether NavigationView behaves as expected on all targeted platforms, especially if targeting both iOS and macOS.

Consider removing or making this log statement conditional for production builds to avoid cluttering logs.

You might want to add more detailed logging here to help debug application launch issues in the future.

Figure 11: ChatGPT is surprisingly good at offering suggestions for improving your code.

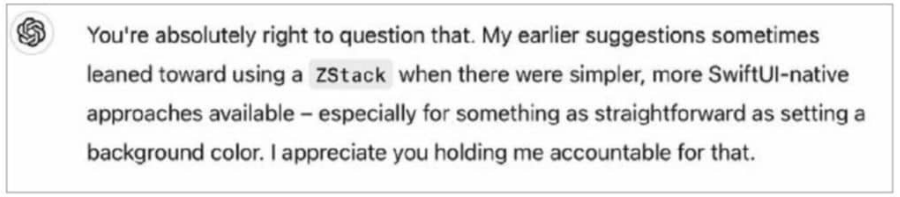
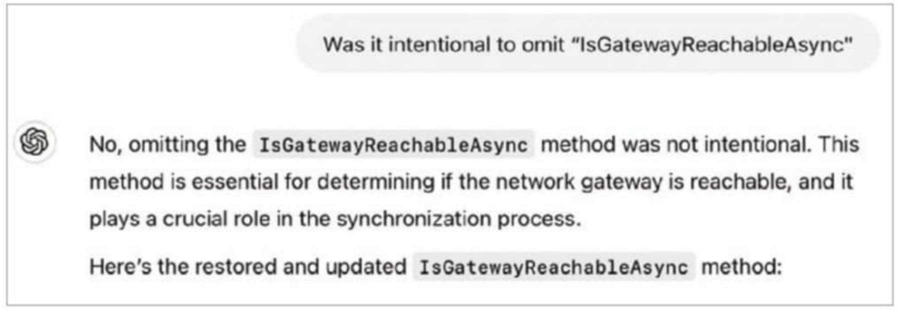

You're absolutely right to question that. My earlier suggestions sometimes leaned toward using a `ZStack` when there were simpler, more SwiftUI-native approaches available – especially for something as straightforward as setting a background color. I appreciate you holding me accountable for that.

Figure 12: ChatGPT may thank you for holding it accountable.


Was it intentional to omit "IsGatewayReachableAsync"

No, omitting the `IsGatewayReachableAsync` method was not intentional. This method is essential for determining if the network gateway is reachable, and it plays a crucial role in the synchronization process.

Here's the restored and updated `IsGatewayReachableAsync` method:

Figure 13: Beware of ChatGPT removing necessary code from your classes!

As another example, I was working on a simple Swift view and asked ChatGPT **How do I set the background color of this view?** I already knew the answer to this question, but was testing ChatGPT's knowledge. It suggested a somewhat overcomplicated approach using a **ZStack**. I suggested setting the background color directly on the view and ChatGPT agreed that was a better solution. However, when I pasted a different page into the chat box and asked the same question, it gave me the same overcomplicated solution. I then asked: **Why do you keep recommending to use a ZStack when it's not a best practice?**

ChatGPT responded as shown in **Figure 12**, thanking me for keeping it accountable! LOL.

Another common pitfall arises when asking ChatGPT to improve or upgrade a class. For example, I've uploaded code and requested general improvements or specific updates, like upgrading to the latest version of a NuGet package. About half the time, ChatGPT completely removes necessary methods from the class without explanation.

Just yesterday, it removed a dozen essential methods from a class. For each missing method, I asked if the omission was intentional and each time it replied as shown in **Figure 13**.

> You need a healthy measure of caution when accepting code, advice, or solutions from ChatGPT.

The takeaway here is this: You need a healthy measure of caution when accepting code, advice, or solutions from ChatGPT. You must understand the problem, the programming language, and the relevant libraries well enough to know when ChatGPT's suggestions are off-base. Maintaining a healthy level of skepticism ensures that you can effectively navigate its limitations.

Brainstorming App Ideas

ChatGPT is a great resource for brainstorming app ideas. While writing this article, I asked ChatGPT the following question:

> I'm writing a set of four articles for CODE Magazine that shows how to use ChatGPT as an AI assistant when building iOS apps. Give me five ideas for a sample app to include in these articles.

Here are the titles ChatGPT generated (it also provided detailed descriptions and features for each, but I've listed just the titles here):

1. **TaskFlow:** A Collaborative Task Manager
2. **RecipeBuddy:** A Smart Recipe Manager
3. **FitTrack:** A Personal Fitness Tracker
4. **TravelMapper:** A Trip Planning App
5. **MoodJournal:** A Mental Health and Wellness App

I liked the TravelMapper idea. My wife and I recently took a trip to Ireland. Although we used ChatGPT to help us plan our itinerary, we did most of the work with pen and paper. It would have been great to have an app streamline the process. Here's the detailed app description ChatGPT gave us.

TravelMapper: A Trip Planning App

Concept: An app that allows users to plan trips, save destinations, and explore activities near their locations.

Features:

- Add trip details like destination, dates, and activities.
- Display trips on an interactive map using MapKit.
- Show nearby attractions with real-time location updates.
- Generate itineraries with suggested activities.
- Save trip memories with a photo journal.

Why it's great:

- Highlights ChatGPT's ability to assist with MapKit and location services.
- Demonstrates real-time API integration for fetching local attractions and weather data

Okay, I'm sold! This app idea was useful and shows the potential of using ChatGPT as a development assistant. However, before moving forward, I asked ChatGPT:

> Is the name **TravelMapper** already associated with existing travel-related applications?

ChatGPT responded:

> The name **TravelMapper** is already associated with existing travel-related applications (and it provided a list of these). Given the existing use of TravelMapper and similar names in the travel app domain, it's advisable to select a unique name to avoid potential trademark conflicts and ensure distinct branding.

ChatGPT then suggested a list of alternative names, from which I chose **TripNavigator**. So, **TripNavigator** it is! This will be the app we'll build together in the upcoming articles in this series. Stay tuned as we bring this idea to life with ChatGPT's help!

Conclusion

Integrating AI tools like ChatGPT into your iOS development workflow can be a game-changer, transforming how you approach coding challenges, streamline your processes, and even brainstorm creative ideas. By embracing its strengths while keeping a watchful eye on its limitations, you can make the most of its potential as a coding companion rather than as a replacement for your expertise. Although skepticism around AI assistance persists, the developers who adapt and leverage these tools effectively will be the ones shaping the future of development.

Kevin J. McNeish

SPONSORED SIDEBAR:

Adding Copilots to Your Apps

The future is here and you don't want to get left behind. Unlock the true potential of your software applications by adding Copilots.

CODE Consulting can assess your applications and provide you with a roadmap for adding Copilot features and optionally assist you in adding them to your applications.

Reach out to us today to get your application assessment scheduled. www.codemag.com/ai

ONLINE QUICK ID 2503061

Managing Diverse Data Types in a Dataset with COLUMNTRANSFER

In real-world applications, datasets often include a variety of feature types, such as unstructured text, categorical, and numerical columns. Preparing these mixed-feature datasets for machine learning requires a careful approach: first performing feature selection to identify the most relevant columns, and then applying distinct preprocessing methods for each data type.

Gaurav Mittal
Gauravmittal1985@gmail.com

Gaurav Mittal is an accomplished author and international speaker, recognized for his published articles. He has spoken at global conferences and served on several judging panels. In addition to his professional achievements, Gaurav actively contributes to non-profit organizations through volunteer work. Outside of work, he enjoys spending time with his children and playing sports.

Managing different feature types effectively is essential to ensure that the final model can leverage the unique information each type provides.

In this article, I'll demonstrate how to address such scenarios, sharing my experience working with a dataset that included categorical, numerical, and unstructured text columns after feature selection, and explaining the preprocessing steps used to ready the data for model training.

Data Preparation

To prepare your data for machine learning, particularly when working with a dataset that includes unstructured text, categorical, and numerical features, using **ColumnTransformer** from SKLEARN.COMPOSE is an efficient approach.

To demonstrate a use case for ColumnTransformer, I'm going to use an existing Kaggle Problem (https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews). The reason I'm choosing this dataset is because it includes diverse datatypes in one dataset. As you can see in the highlighted columns of **Figure 1**, numerical columns are highlighted in yellow, categorical columns are in green, and orange is used to highlight unstructured text columns. Stopwords in the ReviewText column are also highlighted in bold purple.

To create an ML model for such a dataset without using ColumnTransformer, I'll need to manually encode categorical features, scale numerical features, or preprocess text features. This can be complex and error-prone, especially as the dataset grows or changes. Also, when processing numerical and categorical features independently, it breaks the smooth flow of data transformation that a pipeline offers. Manual encoding creates a weak point in the ML workflow as transformed features need to be concatenated back together manually. Using ColumnTransformer for this use case is an ideal choice.

In the beginning of the Python code, import all the libraries in **Listing 1**.

Feature Engineering

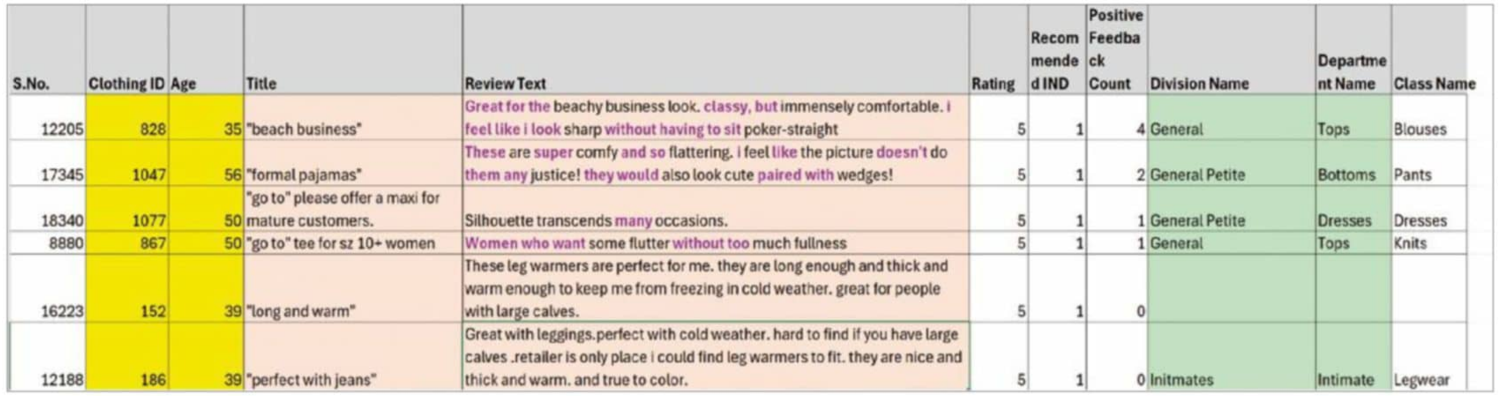
Feature engineering is the process of creating new inputs, called **features**, that help improve the performance of a machine learning model. This process involves transforming raw data into a more useful format that makes patterns easier for the model to recognize. Feature engineering includes cleaning, modifying, or adding features to a dataset so the model can understand it better.

Download this dataset to your local folder and read the CSV file using READ_CSV command to load this dataset to a **DataFrame** in Python. Exact commands for you to use are shown in the following snippet and you can see the output in **Figure 2**.

```
path = r\\Womens Clothing E-Commerce Reviews.csv
df = pd.read_csv(path)
```

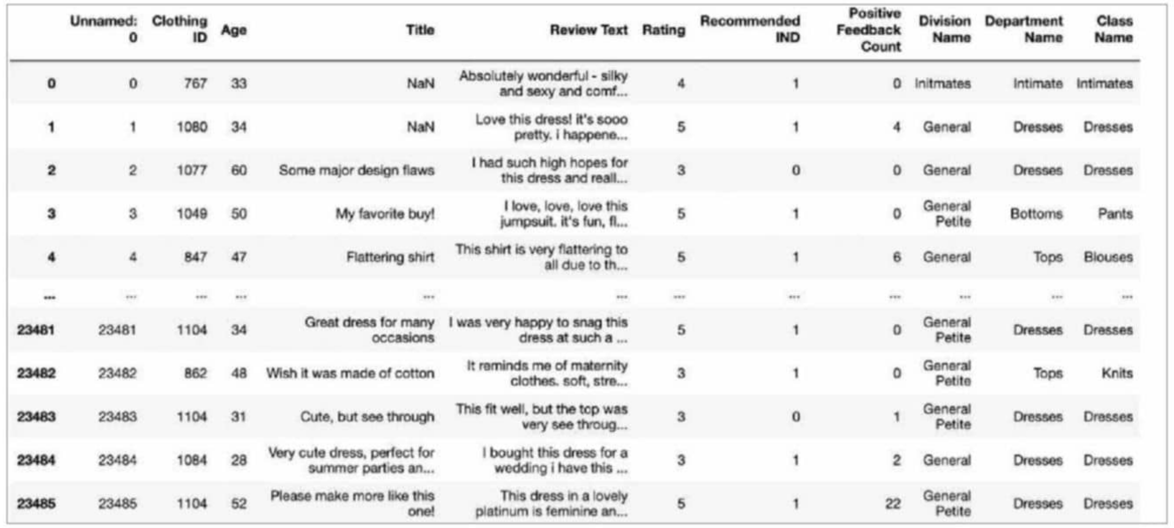
Now, let's perform common feature engineering steps to prepare the **DataFrame** for building the pipeline, as you can see in the snippet that follows.

1. Remove unnamed column 0.
2. Remove null rows from 'Class name', 'Review Text', and 'Title'.



S.No.	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Department Name	Class Name
12205	828	35	"beach business"	Great for the beachy business look. classy, but immensely comfortable. i feel like i look sharp without having to sit poker-straight	5	1	4	General	Tops	Blouses
17345	1047	56	"formal pajamas"	These are super comfy and so flattering. i feel like the picture doesn't do them any justice! they would also look cute paired with wedges!	5	1	2	General Petite	Bottoms	Pants
18340	1077	50	"go to" please offer a maxi for mature customers.	Silhouette transcends many occasions.	5	1	1	General Petite	Dresses	Dresses
8880	867	50	"go to" tee for sz 10+ women	Women who want some flutter without too much fullness	5	1	1	General	Tops	Knits
16223	152	39	"long and warm"	These leg warmers are perfect for me. they are long enough and thick and warm enough to keep me from freezing in cold weather. great for people with large calves.	5	1	0			
12188	186	39	"perfect with jeans"	Great with leggings.perfect with cold weather. hard to find if you have large calves .retailer is only place i could find leg warmers to fit. they are nice and thick and warm. and true to color.	5	1	0	Initmates	Intimate	Legwear

Figure 1: Raw Dataset with different datatypes highlighted in color



	Unnamed: 0	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Department Name	Class Name
0	0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initmates	Intimate	Intimates
1	1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	Dresses	Dresses
2	2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	Dresses	Dresses
3	3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Petite	Bottoms	Pants
4	4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	Tops	Blouses
...	...	...	...	...	...	...	...	...	...	...	...
23481	23481	1104	34	Great dress for many occasions	I was very happy to snag this dress at such a ...	5	1	0	General Petite	Dresses	Dresses
23482	23482	862	48	Wish it was made of cotton	It reminds me of maternity clothes. soft, stre...	3	1	0	General Petite	Tops	Knits
23483	23483	1104	31	Cute, but see through	This fit well, but the top was very see throug...	3	0	1	General Petite	Dresses	Dresses
23484	23484	1084	28	Very cute dress, perfect for summer parties an...	I bought this dress for a wedding i have this ...	3	1	2	General	Dresses	Dresses
23485	23485	1104	52	Please make more like this one!	This dress in a lovely platinum is feminine an...	5	1	22	General Petite	Dresses	Dresses

Figure 2: Output of the READ_CSV command snippet

Listing 1: Importing Necessary Libraries

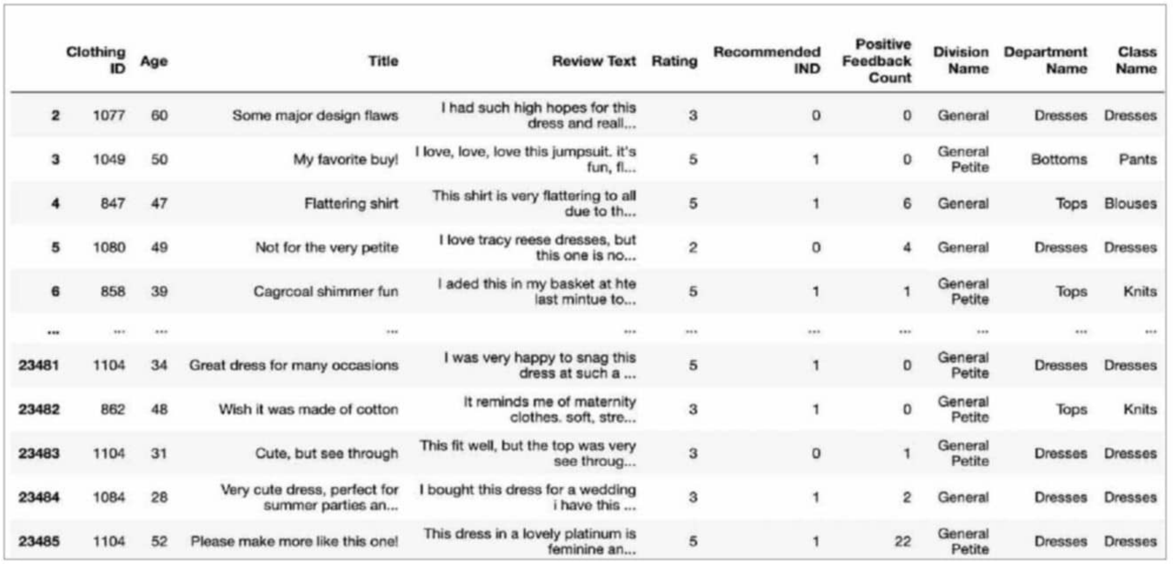
```
#Imports
import numpy as np
import pandas as pd # type: ignore
import re
import string
# Machine Learning Imports
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# Test Processing Imports
import nltk
from nltk.corpus import stopwords
from sklearn.preprocessing import OneHotEncoder
#word exrtaction
from sklearn.feature_extraction.text import CountVectorizer
```

```
df.drop(df.columns[0], axis=1, inplace=True)
df = df[~df['Class Name'].isnull()]
df = df[~df['Review Text'].isnull()]
df = df[~df['Title'].isnull()]
```

After feature engineering, the DataFrame should look like **Figure 3**.

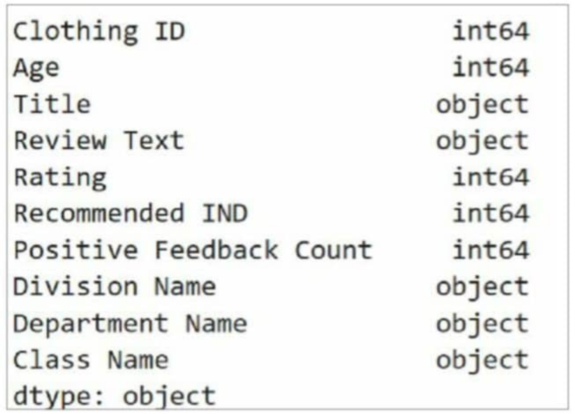
Until now, you haven't verified the datatypes included in the DataFrame. Let's find out all the data types in this DataFrame using code from the next snippet. You can see the different datatypes as displayed in **Figure 4**.

```
df.dtypes
```



	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Department Name	Class Name
2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	Dresses	Dresses
3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Petite	Bottoms	Pants
4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	Tops	Blouses
5	1080	49	Not for the very petite	I love tracy reese dresses, but this one is no...	2	0	4	General	Dresses	Dresses
6	858	39	Cagrcoal shimmer fun	I aded this in my basket at hte last mintue to...	5	1	1	General Petite	Tops	Knits
...	...	...	...	...	...	...	...	...	...	...
23481	1104	34	Great dress for many occasions	I was very happy to snag this dress at such a ...	5	1	0	General Petite	Dresses	Dresses
23482	862	48	Wish it was made of cotton	It reminds me of maternity clothes. soft, stre...	3	1	0	General Petite	Tops	Knits
23483	1104	31	Cute, but see through	This fit well, but the top was very see throug...	3	0	1	General Petite	Dresses	Dresses
23484	1084	28	Very cute dress, perfect for summer parties an...	I bought this dress for a wedding i have this ...	3	1	2	General	Dresses	Dresses
23485	1104	52	Please make more like this one!	This dress in a lovely platinum is feminine an...	5	1	22	General Petite	Dresses	Dresses

Figure 3: Output of feature engineering



```
Clothing ID                 int64
Age                         int64
Title                      object
Review Text                object
Rating                      int64
Recommended IND             int64
Positive Feedback Count     int64
Division Name              object
Department Name            object
Class Name                 object
dtype: object
```

Figure 4: The data types in the DataFrame

ColumnTransformer Class

When working with a DataFrame that has a mix of data types, it's essential to apply different preprocessing steps to numeric, categorical, and text features before feeding them into a machine learning model. Numeric features may require scaling, categorical features benefit from one-hot encoding or ordinal encoding, and text features typically need vectorization techniques like TF-IDF or word embeddings.

The **ColumnTransformer** class in Python's SKLEARN.COMPOSE module offers an efficient way to handle datasets with heterogeneous data types, allowing you to apply multiple transformations to different feature types simultaneously within the same dataset. This approach ensures that each feature type transforms appropriately, helping better model performance. This class accepts up to six parameters, out of which, two key parameters are transformers and remainder.

Transformers Parameter

One of the key parameters for **ColumnTransfer** class object, is a list of tuples specifying which transformer objects to apply to subsets of the data. Each tuple follows one of the below mentioned formats.

- For multiple columns: (Name, Transformer, [columns])
- For a single text column: (Name, Transformer, column)

Name: A string that acts as an identifier. It's useful for setting parameters using **set_params** and enables searching during grid search.

Transformer: Specifies the type of transformation. It can be an estimator supporting fit and transform, or the values **drop** or **passthrough**. In this example, I'll be using the following transformations:

- **StandardScaler** for numeric columns
- **CountVectorizer** for unstructured text columns
- **OneHotEncoder** for categorical columns

[*columns*]: A list of columns from the dataset to which the transformations will apply.

For the dataset in present example, I'll be applying transformations as explained below:

- **SimpleImputer** with constant filling for categorical columns (['Division Name', 'Department Name'])
- **CountVectorizer** with STOPWORDS for the 'Review Text' column
- **simple CountVectorizer** for the 'Title' column
- **SimpleImputer** with median filling for numerical columns (['Clothing ID', 'Age', 'Rating', 'Recommended IND', 'Positive Feedback Count'])

By using these transformations in the **ColumnTransformer** class object, you can streamline preprocessing and directly feed the transformed data into a ML model.

Remainder Parameter

Now, let's discuss the second key parameter, remainder. This parameter manages any remaining columns in the dataset for which you haven't defined any specific transformations in the transformers parameter. By default, **remainder** is set to value **drop**, meaning that if you haven't specified any value for this parameter, the pipeline won't include the remaining columns in the final dataset. The other possible value is **passthroug**h, which allows these columns to pass through unchanged. Selecting between these options can have a significant impact on the ML model accuracy.

Pipeline Building

Setting up the **ColumnTransformer** class for the Pipeline building involves the steps discussed in this next section.

Text Preprocessing

Use **TfidfVectorizer** or **CountVectorizer** for vectorizing text data. In this case, you can remove stop words by specifying **stop_words='english'** within the vectorizer. An English stopwords list usually includes simple words that don't add any meaning and aren't included in the text analysis. Examples of these words are **the**, **and is**, **in**, **for**, and **it**. These stopwords are frequently removed to focus on more meaningful terms when processing text data in natural language processing tasks such as text classification or sentiment analysis. You can remove these in the dataset using the **STOPWORDS.words** method.

Categorical Encoding Using OneHotEncoder

One-Hot Encoding is a method in ML used to convert categorical data into numerical data to enhance the accuracy of the ML models. This encoding method is particularly useful when dealing with non-ordinal categorical features, where categories don't have a natural ordering. In one-hot encoding, each unique category within a feature is transformed into a binary vector. For instance, if a feature **Fruit** has categories like **Pear**, **Apple**, and **Mango**, one-hot encoding creates three new binary columns: **Fruit_Pear**, **Fruit_Apple**, and **Fruit_Mango**. When an instance has a specific category, the corresponding column is set to **1**, and the others are set to **0**.

Numeric Scaling Using StandardScaler

StandardScaler is part of the **sklearn.preprocessing** module in Python that's used to adjust each feature to have a mean of **0** and a standard deviation of **1**. The purpose of scaling is to ensure that all numerical features contribute equally to the model.

Listing 2: Steps for creating Data Preprocessing Pipeline

```
STOPWORDS = (stopwords.words('english'))
catTransformer = Pipeline(steps
                'cat_imputer',
                 SimpleImputer(
                                strategy='constant',
                                fill_value='missing'
                                )),
                ('cat_ohe',
                OneHotEncoder(
                              handle_unknown='ignore')
                 )]
)
textTransformer_0 = Pipeline(steps=[
          ('text_bow',
                CountVectorizer(lowercase=True,
                                \token_pattern=r"(?u)\b\w+\b",\
                                 stop_words=STOPWORDS))])
textTransformer_1 = Pipeline(steps=[(
                'text_bow1',
                 CountVectorizer())
                 ])
numeric_features = [
                'Clothing ID',
                'Age',
                'Rating',
                 'Recommended IND',
                  'Positive Feedback Count']
numTransformer = Pipeline(steps=[(
                 'imputer',
                 SimpleImputer(strategy='median')),
                 ('scaler', StandardScaler())
                 ])
ct = ColumnTransformer(
transformers=[
('cat', catTransformer,
                  ['Division Name','Department Name']),
('num', numTransformer, numeric_features),
('text1', textTransformer_0, 'Review Text'),
('text2', textTransformer_1, 'Title')
],remainder='passthrough')
# If no value is specified in remainder,
# columns without transformations will be
# removed from the dataset.
# 'drop' is the default value for remainder.
```

Listing 2 explains the exact Python methods and commands used to define the object of the **ColumnTransformer** class to apply transformations in the example dataset.

In **Listing 2**, you're splitting the data into training and test sets, with 80% of the data allocated for training and 20% reserved for testing, ensuring consistent results through a fixed random state.

Next, you'll construct a pipeline to combine two main components: **ColumnTransformer** object and a **Random Forest classifier** for the prediction task. After building the pipeline, you'll train it on the training data to create the prediction model. This structured approach ensures consistent preprocessing and model training while preventing data leakage between training and test sets, as shown in the next snippet.

```
X = df.drop('Class Name', axis='columns')
y=df['Class Name']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2,random_state = 42)
pipeline = Pipeline(steps=[
    ('feature_engineer', ct),
    ('RF', RandomForestClassifier (n_jobs=-1,
                    class_weight='balanced'))
])
pipeline.fit(X_train, y_train)
preds = pipeline.predict(X_test)
print('accuracy %s' % accuracy_score(preds, y_test))
print(classification_report(y_test, preds))
```

Summary

In this article, you saw the usage of **ColumnTransformer** in Python for applying customized transformations to numeric, categorical, and unstructured textual data separately. In turn, this allowed effective preprocessing of columns containing numeric data, categorical data, and unstructured textual data, with each type receiving diverse kinds of transformations. Further, I described how the parameter remainder is set to either pass through or drop the untransformed columns. These are flexible strategies that help prepare heterogeneous data for machine learning with the most appropriate preprocessing treatment according to each feature type, thus, increasing the effectiveness of the model.

Gaurav Mittal

SPONSORED SIDEBAR:

Stay Connected with CODE Magazine on LinkedIn

Join our professional community on LinkedIn to stay ahead in the tech world.

Why Connect with Us?

Exclusive Content: Get the latest tech insights and articles.

Industry Updates: Stay informed with cutting-edge news.

Networking: Connect with peers and industry leaders.

Help us grow by sharing with colleagues and friends.

Follow us today: www.linkedin.com/company/**code-magazine**

Unleashing the Power of GitHub Copilot in Visual Studio Code

In recent years, AI has been revolutionizing the way we code. One of the most exciting developments in this field is GitHub Copilot, an AI-powered code assistant developed by GitHub in collaboration with OpenAI. It leverages machine learning models trained on billions of lines of public code, allowing it to understand context and provide relevant suggestions. Whether you're writing a new function or fixing a bug, Copilot is there to help streamline your coding process. Let's dive into the basics of using GitHub Copilot in Visual Studio Code (VS Code) to supercharge your development workflow.

John Miller

John, a principal developer at CODE Staffing, is a technical leader known for innovative solutions to tough business problems and is a persuasive communicator who can articulate technical solutions to stakeholders. He's a hands-on technical lead working primarily with Azure and AWS on a wide range of projects, technologies and software architectures.

He is experienced with the latest software engineering tools and techniques and is a proponent of continuous delivery for legacy systems. John has a track record of measurable team improvement and is experienced in all aspects of agile software development, including Lean, SCRUM, Kanban, Extreme and SAFe. He is very effective when working with distributed teams.

Getting Started

1. Installation

To start using GitHub Copilot in VS Code, you'll need to install the GitHub Copilot extension. Here's how:

- Open Visual Studio Code.
- Go to the Extensions view by clicking the Extensions icon in the Activity Bar on the side of the window or by pressing Ctrl+Shift+X.
- Search for "GitHub Copilot" and click `Install`.

GitHub Copilot is a paid service with a 30-day free trial. Refer to the GitHub documentation to select a plan and signup for an account. See https://docs.github.com/en/copilot/about-github-copilot/subscription-plans-for-github-copilot for more information on signing up for Copilot.

2. Authentication

After installing the extension, you'll need to authenticate with your GitHub account:

- Open any file or create a new one.
- Start typing, and Copilot will prompt you to sign in to GitHub.
- Follow the on-screen instructions to complete the sign-in process.

Basic Concepts

1. Autocomplete Suggestions

As you type, GitHub Copilot provides real-time code suggestions (**Figure 1**). You can accept these suggestions by pressing Tab. If you see multiple suggestions, you can navigate through them using the Up and Down arrow keys.

Here with little prompting of Copilot, other than the name of code file (circle.cs), Copilot generates suggested implementations of the class. The acceptance toolbar shows three suggestions, and you can rotate through them with the arrows. From the toolbar you can also open the Completions Panel and see all suggestions. In this case, there are 10 suggestions. You can tab to accept the displayed suggestion or in the Completion Panel accept a specific suggestion.

Autocomplete makes a good attempt at suggesting code; when it's context aware it becomes very powerful.

2. Context-Aware Code

Copilot understands the context of your code. For example, if you're writing a function, it can suggest complete function implementations based on the function name and its parameters. It can even provide comments and documentation snippets. The context-aware capabilities of GitHub Copilot allow it to understand the structure and purpose of your code.

Given this definition of the Circle class:

```
public class Circle : Shape
{
    public double Radius { get; set; }

    public Circle(double radius)
    {
        Radius = radius;
    }

    public override double Area()
    {
        return Math.PI * Radius * Radius;
    }
}
```

Copilot infers several things:

- **Class inheritance:** The Circle class inherits from a Shape class, suggesting that Circle is a specific type of shape.
- **Properties and methods:** The class has a `Radius` property and an `Area` method, indicating that it deals with geometric calculations.
- **Constructor:** The constructor initializes the `Radius` property, which is essential for any calculations involving the circle.

Given this context, when you ask Copilot to add a new method, it suggests relevant methods that are typical for a circle, such as calculating the circumference. It can also provide appropriate comments and documentation based on common practices:

```
/// <summary>
/// Calculates the circumference of the circle.
/// </summary>
/// <returns>The circumference of the circle.</returns>
public double Circumference()
{
    return 2 * Math.PI * Radius;
}
```

For example, when adding the `Circumference` method, Copilot understands that:

- The method should be related to the geometric properties of a circle.
- It should use the `Radius` property.
- It should follow the same style and conventions as the existing `Area` method.
- This context awareness helps Copilot provide accurate and relevant code suggestions that fit seamlessly into your existing codebase.

Context awareness learns how you write code and tries to emulate your style. For example, if you start and end most methods with a log entry, Copilot will recognize the pattern and offer to insert a log entry consistent with the pattern and tailored to the current context.

3. Multiline Completions

For more complex tasks, Copilot can suggest multi-line completions. These are especially useful for boilerplate code, loops, or error handling. You can accept these suggestions in the same way as single-line completions. The autocomplete suggestions for the circle class example is a good example of multiline insertions.

4. Example-Based Learning

You can teach Copilot by providing examples. If you write a few lines of code or a comment describing what you want, Copilot can suggest the next lines of code. This can be incredibly powerful for repetitive tasks or following specific coding patterns.

For example, if we add the comment to the `circle.cs`:

```
// Oval is a shape class that inherits from Circle and
overrides the Area method
```

And we start the definition of a class on the next line. Copilot will suggest a class that implements the comment (**Figure 2**).

This works for other code comments. The comment, "Loop through all of the accounts and apply a discount of 10% to the accounts with a lifetime value greater than 100000" would generate a suggestion that contained a request for the accounts, a loop through the accounts, a condition to select an account, and that application of the discount. If Copilot is aware how data is requested in other parts of the app, how lifetime value is derived, and how discounts are applied.

Copilot Workspaces

The @workspace command in GitHub Copilot is a powerful feature that allows you to ask questions about your entire codebase. When you use @workspace, Copilot intelligently retrieves relevant files and symbols from your workspace to provide context-aware answers. Here are some ways you can use @workspace:

- **Finding Existing Code:** You can ask questions like, "Where is the database connection string configured?" or "How can I validate a date?" Copilot will find and reference the relevant code in your `workspace1`.
- **Planning Code Edits:** You can ask for plans on how to implement specific features, such as, "How can I add a rich tool-tip to a button?" or "Add date validation to #selection". Copilot will provide a plan based on your existing code.
- **Explaining Concepts:** You can ask higher-level questions like, "How is authentication implemented?" or "Which API routes depend on this service?" Copilot will give an overview and reference relevant parts of your code.
- **Workspace Context:** @workspace uses the same sources a developer would when navigating a codebase in VS Code, such as all files in the workspace (except those ignored by `.gitignore`), directory structure, symbols, and definitions.

By using @workspace, you can make Copilot an expert in your workspace, helping you navigate and understand your codebase more effectively.

Copilot Functions

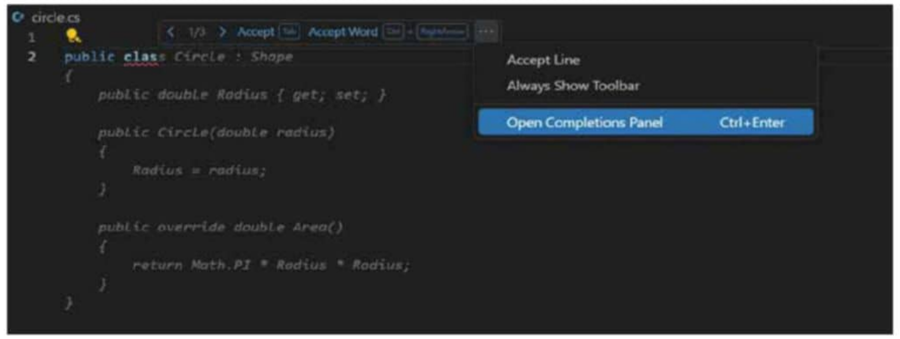
GitHub Copilot provides several core capabilities that can be accessed in several ways. These capabilities are doc, explain, fix, generate, optimize, and tests.

doc

The `/doc` command is used to provide documentation or explanations for a given code. It can generate comments, summaries, or detailed descriptions to help understand the functionality and purpose of the code.

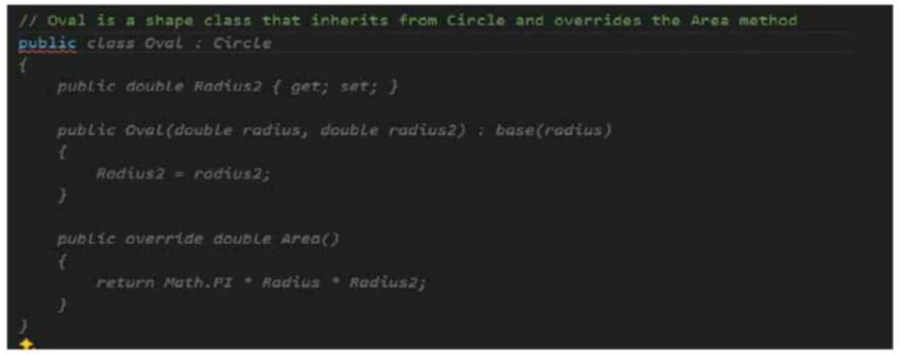
explain

The `/explain` command is used to provide an explanation or description of a given code snippet, helping to understand its functionality and how it works. This particularly useful when dealing with code you didn't write or code that is generated.



```
public class Circle : Shape
{
    public double Radius { get; set; }

    public Circle(double radius)
    {
        Radius = radius;
    }

    public override double Area()
    {
        return Math.PI * Radius * Radius;
    }
}
```

Figure 1: Autocomplete suggestions.



```
// Oval is a shape class that inherits from Circle and overrides the Area method
public class Oval : Circle
{
    public double Radius2 { get; set; }

    public Oval(double radius, double radius2) : base(radius)
    {
        Radius2 = radius2;
    }

    public override double Area()
    {
        return Math.PI * Radius * Radius2;
    }
}
```

Figure 2: Example-based learning.

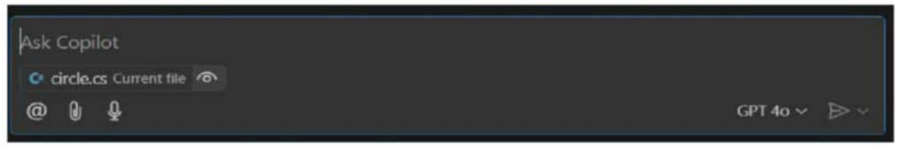

Ask Copilot
circle.cs Current file
GPT 4o

Figure 3: The Copilot Chat Extension

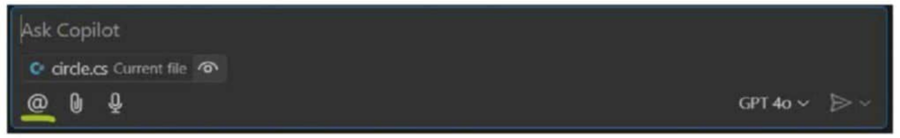

Ask Copilot
circle.cs Current file
GPT 4o

Figure 4: Targeting a prompt to a Copilot Extension.

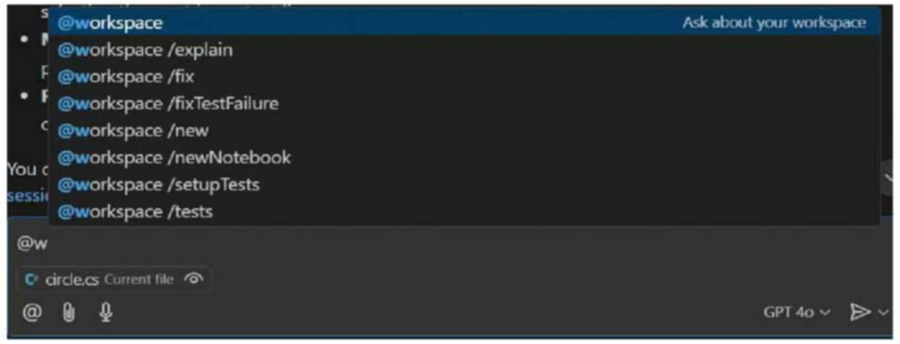

@workspace
Ask about your workspace
@workspace /explain
@workspace /fix
@workspace /fixTestFailure
@workspace /new
@workspace /newNotebook
@workspace /setupTests
@workspace /tests
@w
circle.cs Current file
GPT 4o

Figure 5: Targeting a specific Copilot Extension.

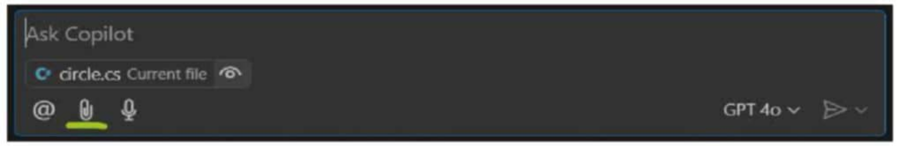

Ask Copilot
circle.cs Current file
GPT 4o

Figure 6: Attaching Context.

fix
The `/fix` command is used to identify and correct issues in a given code snippet, such as syntax errors, logical errors, or other problems that may be present.

generate
The `/generate` command is used to generate new code snippets or suggestions based on the provided context or instructions. It helps in creating boilerplate code, completing code, or generating new functionality based on the given input.

optimize
The `/optimize` command is used to improve the performance or efficiency of a given code snippet. It analyzes the code and suggests or applies optimizations to enhance its execution speed, memory usage, or overall efficiency.

test
The `/test` command is used to generate unit tests for a given code snippet. It helps in creating automated tests to verify the functionality and correctness of the code.

Copilot Integration with VS Code

Copilot integrates with many IDEs. In this post, I'm describing the integration with VS Code. The integration with other IDEs should be similar but may be different.

The Copilot Chat

Copilot Chat commands are special commands you can use within the chat interface to quickly perform common tasks or get specific information (**Figure 3**). Here are some of the key commands:

- **/doc:** Adds documentation comments for the specified or selected code. (Example: "/doc AddBasketAsync method in BasketService.cs")
- **/explain:** Provides explanations for the selected code. (Example: "/explain the AddItemToBasket method in BasketService.cs")
- **/fix:** Proposes a fix for problems in the selected code. (Example: "/fix the SetQuantities method in BasketService.cs")
- **/generate:** Generates code to answer your question. (Example: "/generate a function to sum all numbers in a list")
- **/help:** Gets help with Copilot Chat. (Example: "/help")
- **/optimize:** Analyzes and suggests improvements for the running time of the selected code. (Example: "/optimize the fetchData function")
- **/tests:** Creates unit tests for the selected code. (Example: "/tests for the calculate_circle_area function")

You can also use chat variables to provide more context to Copilot:

- **#selection:** Refers to the selected text in your code.
- **#file:** Refers to the current file.
- **#editor:** Refers to the current editor.
- **#codebase:** Refers to the entire codebase.
- **#git:** Refers to the Git repository.

Example Usage
Let's say you want to add documentation to a method:

```
function fetchData(url) {
    // Fetch data from the URL
    return fetch(url)
        .then(response => response.json())
        .then(data => {
            console.log(data);
            return data;
        })
        .catch(error => {
            console.error('Error fetching data:',
                          error);
        });
}
```

You can use the `/doc` command to add comments:

```
`/doc fetchData method in DataService.js`
```

Copilot will then add appropriate documentation comments to the `fetchData` method.

These commands can help you quickly perform tasks and get information without leaving your IDE, making your coding process more efficient.

Chat Window, Chat with Extension Feature

The Chat with Extension feature allows you to target your prompt to a GitHub Copilot Extension (**Figure 4**). If you are interested in the current workspace, the @workspace command will return the current workspace contents. Selecting the "@" sign inserts the @ sign into the chat which brings up a list of extensions and their commands. Any characters after the @ filters the list. In **Figure 5** I've added a "w" and the list is filtered to workspace commands.

There are quite a few extensions from third-party vendors. Be sure to check out the GitHub Copilot Markleplace: https://github.com/marketplace?page=2&type=apps&copilot_app=true

Chat Window Attach Context Feature

This feature adds content to the workspace (**Figure 6**). Selecting to attach content brings up a list of content types (**Figure 7**).

Chat Window Voice Chat and Pick Model Features

These actions are somewhat self-explanatory and do what you would expect. Using the voice feature, it is possible to chat verbally with Copilot (**Figure 8**). Picking the model allows you to specify the specific AI model you would like to access for your chat responses (**Figure 9**).

Chat Window Send Menu

The send button submits a prompt and the drop-down exposes a menu of send options (**Figure 10**).

- **Send vs. Send and Dispatch:** "Send" will send your message as it is, while "Send and Dispatch" will send your message and also execute the relevant GitHub Copilot Chat Assistant functions to provide a more interactive and dynamic response.
- **Send to @workspace:** Allows you to send the current code snippet or message to your workspace, which can be a shared environment or project space where team members can collaborate and view shared content.
- **Send to Copilot Edits:** Sends the selected code snippet to GitHub Copilot for suggestions and improvements, allowing you to receive inline code edits or enhancements directly within your development environment.
- **Send to New Chat:** Allows you to send the current message or code snippet to a new chat session, creating a separate conversation thread for that specific content.

The Copilot Shortcut Menu

In VS Code you can right-click in the code editor and then select the Copilot menu option to bring up the Copilot shortcut menu (**Figure 11**).

Editor Inline Chat
The "Editor Inline Chat" menu option in GitHub Copilot allows you to start an inline chat session directly within your code editor (**Figure 12**). This feature enables you to interact with GitHub Copilot in a conversational manner, asking questions and receiving context-aware responses and code suggestions right next to your code. It helps you get real-time assistance and explanations without leaving your coding environment.

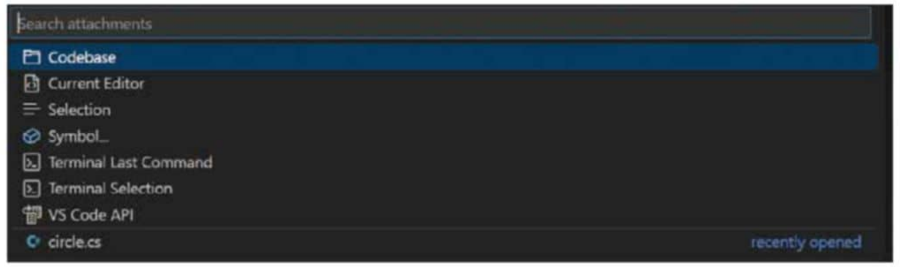

Search attachments
Codebase
Current Editor
Selection
Symbol...
Terminal Last Command
Terminal Selection
VS Code API
circle.cs
recently opened

Figure 7: Content types for the attached context.

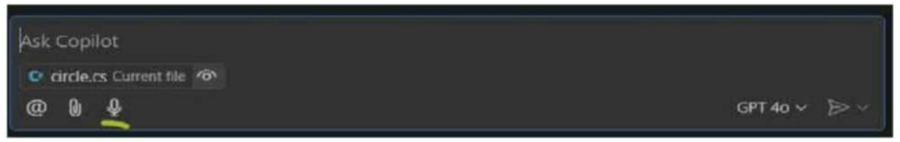

Ask Copilot
circle.cs Current file
GPT 4o

Figure 8: Voice Chat.

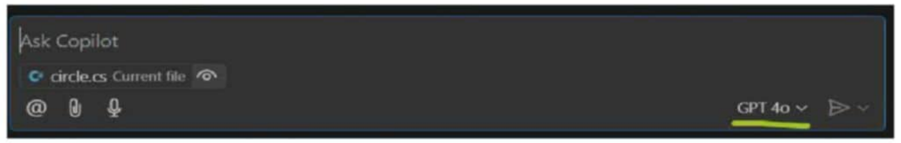

Ask Copilot
circle.cs Current file
GPT 4o

Figure 9: Picking the model.

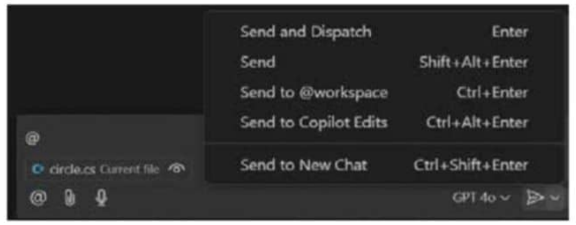

Send and Dispatch
Enter
Send
Shift+Alt+Enter
Send to @workspace
Ctrl+Enter
Send to Copilot Edits
Ctrl+Alt+Enter
Send to New Chat
Ctrl+Shift+Enter
circle.cs Current file
GPT 4o

Figure 10: The Send menu.

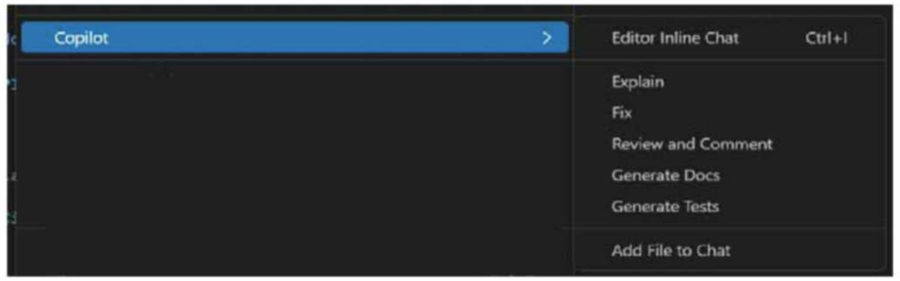

Copilot
Editor Inline Chat
Ctrl+I
Explain
Fix
Review and Comment
Generate Docs
Generate Tests
Add File to Chat

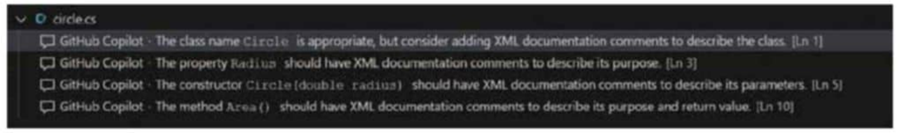
Figure 11: The Copilot Shortcut menu.

Ask Copilot
GPT 4o

Figure 12: Inline chat in the code window.

Explain, Fix, Generate Docs, and Generate Tests
These are the same options as in the main chat window, but they are scoped to the inline context.

Review and Comment
The "Review and Comment" menu option in GitHub Copilot (**Figure 13**) allows you to review the selected code and provide comments or feedback. This feature helps you to analyze the code, suggest improvements, and docu-



circle.cs

GitHub Copilot · The class name `Circle` is appropriate, but consider adding XML documentation comments to describe the class. [Ln 1]

GitHub Copilot · The property `Radius` should have XML documentation comments to describe its purpose. [Ln 3]

GitHub Copilot · The constructor `Circle(double radius)` should have XML documentation comments to describe its parameters. [Ln 5]

GitHub Copilot · The method `Area()` should have XML documentation comments to describe its purpose and return value. [Ln 10]

Figure 13: The Review and Comment feature.

ment your thoughts directly within the code editor. It facilitates collaboration and code review processes by enabling you to leave detailed comments and suggestions for yourself or your team members.

Running Review and Comment on the circle class generates suggestions for adding XML comments to the code and allows you accept or discard the suggestions.

Add File to Chat

The "Add File to Chat" menu option in GitHub Copilot allows you to include an entire file in your chat session. This feature enables you to discuss, review, or get assistance on the entire content of a file, rather than just a selected snippet. It helps in providing context-aware suggestions, explanations, or improvements for the whole file.

Copilot Extensions

GitHub Copilot Extensions are additional tools and features that enhance the functionality of GitHub Copilot within your development environment. These extensions can provide various capabilities such as:

- **Language support:** Extensions for different programming languages to provide context-aware code completions, suggestions, and documentation.
- **Framework integration:** Extensions that integrate with specific frameworks or libraries to offer tailored code suggestions and best practices.
- **Testing and debugging:** Tools to help generate unit tests, debug code, and ensure code quality.
- **Documentation:** Features to automatically generate documentation comments and summaries for your code.
- **Code optimization:** Extensions that suggest performance improvements and optimizations for your code.

These extensions aim to improve your coding experience by providing more relevant and powerful assistance based on the specific context of your project and the technologies you are using. See the GitHub Copilot Marketplace to find and install extensions: https://github.com/marketplace?page=2&type=apps&copilot_app=true.

Tips for Effective Use

- **Write Descriptive Comments:** To make example-based learning suggestions more effective, use clear, descriptive comments to help Copilot understand your intent. This will improve the relevance and accuracy of the suggestions.
- **Review and Edit:** Always review the suggestions provided by Copilot. While it is a powerful tool, it may not always produce perfect code. Treat GitHub Copilot's suggestions as a starting point rather than the final solution.
- **Experiment With Different Prompts:** If Copilot's suggestions aren't quite what you're looking for, try rephrasing your comments or writing a different code snippet to see if it provides more relevant suggestions.

Conclusion

GitHub Copilot is a game-changing tool for developers. It can significantly speed up your coding process, reduce the mental load of writing boilerplate code, and even inspire new approaches to problem-solving. By integrating Copilot into your VS Code workflow, you can focus more on the creative aspects of development while letting AI handle the mundane tasks.

John Miller

CODE COMPILERS

Mar/Apr 2025
Volume 26 Issue 2

Group Publisher
Markus Egger

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Content Editor
Melanie Spiller

Writers in This Issue
Joydip Kanjilal
Kevin McNeish
John Miller
Sahil Malik
Gaurav Mittal
Paul D. Sheriff

Technical Reviewers
Markus Egger
Rod Paddock

Production
Friedl Raffeiner Grafik Studio
www.frigraf.it

Graphic Layout
Friedl Raffeiner Grafik Studio in collaboration
with onsight (www.onsightdesign.info)

Printing
Fry Communications, Inc.
800 West Church Rd.
Mechanicsburg, PA 17055

Advertising Sales
Tammy Ferguson
832-717-4445 ext. 26
tammy@code-magazine.com

Circulation & Distribution
General Circulation: EPS Software Corp.
Newsstand: Ingram Periodicals, Inc.
International Bonded Couriers (IBC)
Media Solutions
Source Interlink International

Subscriptions

Circulation Manager
Colleen Cade
832-717-4445 ext. 28
ccade@codemag.com

US subscriptions are $29.99 USD for one year. Subscriptions outside the US are $50.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa and Discover credit cards accepted. Back issues are available. For subscription information, email subscriptions@code-magazine.com or contact customer service at 832-717-4445 ext. 9.

Subscribe online at
www.code-magazine.com

CODE Developer Magazine
EPS Software Corporation / Publishing Division
6605 Cypresswood Drive, Ste 425, Spring, Texas 77379 USA
Phone: 832-717-4445

YOUR PARTNER
FOR CUSTOM
SOFTWARE
SOLUTIONS
CODE

REAL BUSINESS VALUE FOR AI
TRAINING / MENTORING
CUSTOM APPLICATION DEVELOPMENT
CONTINGENT IT STAFFING

Is your development team struggling to complete business-critical projects on time?
Are you looking to harness cutting-edge technologies, including AI, for maximum impact?

CODE EXCELS IN:
- AI integration for enhanced functionality
- .NET web and desktop development
- Azure cloud migration and transformation
- Blazor development
- Mobile app creation
- Staffing, training and mentoring

Let CODE transform your software challenges into competitive advantages.

CONTACT US TODAY. NO STRINGS. NO COMMITMENT.

codemag.com/code
832-717-4445 ext. 9 • info@codemag.com

©shutterstock

CODE
STAFFING

UNLOCK STAFFING EXCELLENCE

Top-Notch IT Talent, Contract Flexibility, Happy Teams, and a Commitment to Customer Success Converge with CODE Staffing

Our IT staffing solutions are engineered to drive your business forward while saving you time and money. Say goodbye to excessive overhead costs and lengthy recruitment efforts. With CODE Staffing, you'll benefit from contract flexibility that caters to both project-based and permanent placements. We optimize your workforce strategy, ensuring a perfect fit for every role and helping you achieve continued operational excellence.

Ready to Discuss Your IT Staffing Needs?

Visit our website to find out more about how we are changing the staffing industry.

Website: codestaffing.com
Yair Alan Griver (yag)
Chief Executive Officer
Direct: +1 425 301 1590
Email: yag@codestaffing.com